VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PARALELNÍ ŘEŠENÍ PARCIÁLNÍCH DIFERENCIÁLNÍCH ROVNIC

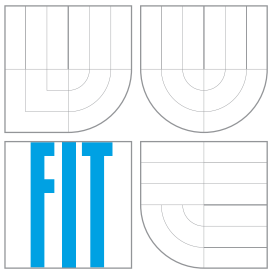
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL ČAMBOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PARALELNÍ ŘEŠENÍ PARCIÁLNÍCH DIFERENCIÁLNÍCH ROVNIC

PARTIAL DIFFERENTIAL EQUATIONS PARALLEL SOLUTIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL ČAMBOR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÁCLAV ŠÁTEK

BRNO 2011

Abstrakt

Práce se zabývá parciálními diferenciálními rovnicemi, pro jejichž řešení je navržen speciální numerický integrátor zpracovávající operandy ve formátu plovoucí řádové čárky. Návrhy jsou postaveny na principech Eulerovy metody i zpracování více členů Taylorovy řady. Práce ukazuje srovnání paralelního a sériového přístupu ke zpracování mantis a exponentů při numerické integraci. V textu najdeme rovněž srovnání specializovaného numerického integrátoru s dostupnými paralelními systémy.

Abstract

This thesis deals with the concepts of numerical integrator using floating point arithmetic for solving partial differential equations. The integrator uses Euler method and Taylor series. Thesis shows parallel and serial approach to computing with exponents and signifi- cands. There is also a comparison between modern parallel systems and the proposed concepts.

Klíčová slova

Parciální diferenciální rovnice, numerická integrace, simulace, aritmetika pevné řádové čárky, aritmetika plovoucí řádové čárky, Taylorova řada.

Keywords

Partial differential equation, numerical integration, simulation, fixed point arithmetic, floating point arithmetic, Taylor series.

Citace

Michal Čambor: Paralelní řešení parciálních
diferenciálních rovnic, diplomová práce, Brno, FIT VUT v Brně, 2011

Paralelní řešení parciálních diferenciálních rovnic

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Václava Šátka

.....
Michal Čambor
24. května 2011

Poděkování

Za odborné vedení mé diplomové práce bych chtěl poděkovat panu Ing. Václavu Šátkovi, který mi svými připomínkami, náměty a radami pomáhal směřovat práci ke zdárnému cíli. Dále bych chtěl poděkovat rodině a přátelům za podporu po celou dobu studia.

Obsah

Kapitola 1

Úvod

Diplomová práce se zabývá paralelním řešením parciálních diferenciálních rovnic. Kapitola 2 se zaměřuje na numerická řešení jednoduchých parciálních diferenciálních rovnic. Parciální diferenciální rovnice se standartně převádějí na obyčejné diferenciální rovnice, což vede na rozsáhlé soustavy. Tyto soustavy obyčejných diferenciálních rovnic se řeší paralelně. Řešení ukážeme pro dva základní problémy a to vedení tepla v tenké tyči (kapitola 2.1.3) a na kmitající struně v kapitole 2.1.4. Řešením nám bude soustava jednoduchých diferenciálních rovnic, které ke svému řešení využijí integrátor. Na návrh tohoto integrátoru se zaměříme v dalších kapitolách.

Pro řešení diferenciálních rovnic nejprve zvolíme numerickou metodu vycházející z prvních dvou členů Taylorova rozvoje zvanou Eulerova metoda. V kapitole 3 ukážeme její jednoduchou obvodovou realizaci, jež bude výchozím schématem pro návrh numerického integrátoru. Před tím se však seznámíme s reprezentováním čísel na počítačích. Základními formáty jsou čísla s pevnou řádovou čárkou a čísla ve formátu plovoucí řádové čárky. Vzhledem k řešení numerické integrace v prvotní fázi za pomoci Eulerovy metody, zaměříme se v rámci těchto formátu na operace sčítání, odečítání a násobení. Tyto poznatky nám budou východiskem k návrhu numerických integrátorů, jejichž základ bude tvořit navržená násobička postavena na principu Boothova násobení (kapitola 3.1.3). V kapitole 3.3 ukážeme a vysvětlíme princip činnosti numerického integrátoru pracující s čísly v pevné řádové čárce. Poznatky z této kapitoly rozšíříme o některé další prvky, které si vynucuje práce s čísly v plovoucí řádové čárce. Ukážeme návrh numerického integrátoru pro práci s čísly ve formátu plovoucí řádové čárky. Přesněji numerického integrátoru zpracovávající mantisu a exponent postupně v jedné jednotce 3.4.2 a v numerickém integrátoru, který je tvořen dvěmi dílčími jednotkami. Jednotkou mantisy (kapitola 3.4.3) a exponentu (kapitola 3.4.3). Závěrem kapitoly navrhneme rozšíření integrátorů pracujících v plovoucí řádové čárce o možnost počítání více členů Taylorova rozvoje (kapitola 3.5). Ukážeme modifikace algoritmů a obvodové schéma obou přístupů. Postupného zpracování mantisy a exponentu (3.5.1) a variantu zpracovávající mantisu a exponent paralelně (3.5.2).

Poznatky z kapitoly 3 využijeme v kapitole 4 k návrhu simulátoru. Přesněji budeme vycházet z numerických integrátorů pracujících na principu Eulerovy metody. Postupně navrhneme koncept simulátoru (4.1), GUI (4.2) a metody pro porovnání (4.3) realizací. Ty nám umožní zaznamenat výsledky (4.5) a provést nad nimi zhodnocení (4.6).

Na kapitoly zabývající se našim řešením naváže kapitola 5. Zaměřuje se na uvedení do problematiky současných paralelních systémů (5.1) a jejich použití (kapitola 5.2). Kapitola poskytuje i zhodnocení těchto paralelních přístupů vzhledem k našemu problému a srovnání s naším přístupem specializovaných numerických integrátorů.

V závěru pak zhodnotíme dosažené výsledky práce.

Kapitola 2

Parciální diferenciální rovnice

Nejprve se seznámíme s pojmem parciální derivace. Parciální derivace je zobecněním pojmu derivace, kdy nepracujeme s jednou proměnnou, ale neznámých máme více [11]. V případě parciální derivace se považuje za proměnnou pouze ta proměnná, která se vyskytuje v derivaci. Ostatní proměnné jsou chápany jako konstanty. Parciální derivace se značí stejně jako obyčejná derivace, ale místo symbolu d je používán symbol ∂. Tedy:

$$\frac{\partial f}{\partial y}$$

značí parciální derivaci funkce f podle proměnné y. Diferencální rovnice je matematická rovnice, ve které vystupují proměnné jako derivace funkcí. Rozlišujeme u nich dva typy a to obyčejné diferenciální rovnice a diferenciální rovnice parciální. Liší se počtem neznámých, které se vyskytují v derivaci. V případě obyčejné diferenciální rovnice se vyskytují derivace hledané funkce jen dle jedné proměnné. Naproti tomu u parciální diferenciální rovnice se vyskytují parciální derivace. Obecně lze parciální diferenciální rovnici zapsat ve tvaru:

$$F(x_1, x_2, ..., x_n, z, \frac{\partial z}{\partial x_1}, ..., \frac{\partial z}{\partial x_n}, \frac{\partial^2 z}{\partial x_1^2}, \frac{\partial^2 z}{\partial x_1 \partial x_2}, ..., \frac{\partial^2 z}{\partial x_1 \partial x_n}, \frac{\partial^2 z}{\partial x_2^2}, ..., \frac{\partial^k z}{\partial x_n^k}, ...) = 0$$

kde $z(x_1, x_2, ..., x_n)$ je neznámá funkce n proměnných. Řád nejvyšší derivace, která se v rovnici vyskytuje, pak určuje i řád rovnice.

2.1 Řešení parciálních diferenciálních rovnic

Řada technických a fyzikálních problémů vede k řešení parciálních diferenciálních rovnic (PDR). Náš zájem bude soustředěn na řešení PDR na elektronických diferenciálních analyzátorech. Přímé řešení PDR na elektronických diferenciálních analyzátorech není možné, protože máme k dispozici pouze jednu nezávislou proměnnou a to je čas. Je tedy nutno zvolit náhradní soustavu rovnic, v nichž jedna proměnná je spojitá (reprezentována strojovým časem) a ostatní proměnné jsou nahrazeny diskrétními přírůstky (tzv. metoda přímek).

V dalších úvahách se budeme zabývat řešením nejznámějších typů PDR:

vlnová rovnice (hyperbolická) $\frac{\partial^2 n}{\partial t^2} = c^2 \frac{\partial^2 n}{\partial x^2}$

difuzní rovnice (parabolická) $\frac{\partial n}{\partial t} = a^2 \frac{\partial^2 n}{\partial x^2}$

Laplaceova rovnice (eliptická) $\frac{\partial^2 n}{\partial x^2} + \frac{\partial^2 n}{\partial y^2} + \frac{\partial^2 n}{\partial z^2} = 0$

Poissonova rovnice (eliptická) $\frac{\partial^2 n}{\partial x^2} + \frac{\partial^2 n}{\partial y^2} + \frac{\partial^2 n}{\partial z^2} = -c$

Pro řešení PDR jsou výhodné diferenční metody. Jsou založené na aproximaci parciálních derivací konečnými diferenčními podíly. Nahradíme-li všechny parciální derivace v rovnici, dostaneme metodu sítí. Když parciální derivace podle jedné proměnné ponecháme spojité a derivace podle ostatních proměnných nahradíme diferenčními podíly, dostaneme metodu přímek. Používá se jí při řešení PDR parabolického nebo hyperbolického typu, protože jednou nezávislou proměnnou je u nich čas. Takže tato úloha má charakter počátečního problému. Bude-li nezávislou proměnnou prostorová veličina, bude řešením okrajový problém.

V literatuře jsou tříděny podle toho, kterou z nezávisle proměnných ponecháme spojitou a kterou diskretizujeme:

a) metoda DSCT (discrete space continous time) - diskrétně prostorové souřadnice a spojitě čas,

b) metoda CSDT (continous space - discrete time) - spojitě prostorové souřadnice a diskrétně čas,

c) metoda DSDT (discrete space - discrete time) - diskrétně prostorové souřadnice i čas.

2.1.1 Metoda přímek

Metodu přímek si odvodíme na příkladě vedení tepla v jednorozměrné ideální tyči. Tento problém je popsán parabolickou rovnicí:

$$\frac{\partial u}{\partial t} = a(x)\frac{\partial^2 u}{\partial x^2}, \qquad 0 < x < L, \qquad t > 0 \tag{2.1}$$

kde $u = u(x,t)$ je teplotní funkce, $a(x)$ je měrná teplotní vodivost. Počáteční podmínka je:

$$u(x,0) = f(x), \qquad 0 < x < L \tag{2.2}$$

Okrajové podmínky:

$$u(0,t) = u_1(t) \qquad \text{a} \qquad u(L,t) = u_2(t), \qquad t > 0 \tag{2.3}$$

Časovou proměnnou t necháme měnit spojitě a prostorovou proměnnou diskrétně. Kdybychom volbu provedli obráceně, nastaly by nám problémy s řešením okrajových podmínek a to i pro derivace.

Interval $< 0, L >$ proměnné x, kterou budeme řešit diskrétně, rozdělíme na n dílků o délce h:

$$\Delta x = h = \frac{L}{n} \tag{2.4}$$

a parciální derivace ve směru x nahradíme v každém diskrétním bodě (tzv. uzlu) podílem konečných diferencí. První parciální derivace $u = u(x,t)$ podle x je obecné:

$$\frac{\partial u(x,t)}{\partial x} = \lim_{\Delta x \to 0} \frac{u(x + \Delta x, t) - u(x,t)}{\Delta x} \tag{2.5}$$

Pro konečnou hodnotu $x = h$ dostaneme první aproximaci, která v případě symetrických diferencí v libovolném bodě intervalu je:

$$\left.\frac{\partial u(x,t)}{\partial x}\right|_x \approx \frac{u(x + h/2, t) - u(x - h/2, t)}{h} \tag{2.6}$$

a nebo v bodě $x = x_i$:

$$\left.\frac{\partial u(x,t)}{\partial x}\right|_{x_i} \approx \frac{u(x_{i+1/2},t) - u(x_{i-1/2},t)}{\Delta x} \approx \frac{u_{i+1/2}(t) - u_{i-1/2}(t)}{\Delta x} \tag{2.7}$$

Můžeme přepsat do tvaru:

$$\left.\frac{\partial u(x,t)}{\partial x}\right|_{x_i} \approx \frac{u_{i+1}(t) - u_{i-1}(t)}{2\Delta x} \tag{2.8}$$

V případě nesymetrických vzorců:

$$\left.\frac{\partial u(x,t)}{\partial x}\right|_{x_{i+1/2}} \approx \frac{u_{i+1}(t) - u_i(t)}{\Delta x} \tag{2.9}$$

$$\left.\frac{\partial u(x,t)}{\partial x}\right|_{x_{i-1/2}} \approx \frac{u_i(t) - u_{i-1}(t)}{\Delta x} \tag{2.10}$$

Na základě odvozených vzorců můžeme uvést aproximace vyšších derivací:
2. parciální derivace:

$$\left.\frac{\partial^2 u(x,t)}{\partial x^2}\right|_{x_i} \approx \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{(\Delta x)^2} \tag{2.11}$$

3. parciální derivace:

$$\left.\frac{\partial^3 u(x,t)}{\partial x^3}\right|_{x_i} \approx \frac{u_{i+2}(t) - 3u_{i+1}(t) + 3u_{i-1}(t) - u_{i-2}(t)}{2(\Delta x)^3} \tag{2.12}$$

4. parciální derivace:

$$\left.\frac{\partial^4 u(x,t)}{\partial x^4}\right|_{x_i} \approx \frac{u_{i+2}(t) - 4u_{i+1}(t) + 6u_i(t) - 4u_{i-1}(t) + u_{i-2}(t)}{2(\Delta x)^4} \tag{2.13}$$

Všeobecně platí při použití metody přímek zásada, že celkovou přesnost řešení je možno zvyšovat buď zvyšováním počtu přímek nebo použitím přesnějších vzorců. První způsob vede ke zvýšení počtu použitých počítacích jednotek, druhý dává komplikovaná a nepřehledná schémata počítací sítě.

Při použití metody přímek je nutné rozhodnout, kterou proměnnou budeme zobrazovat spojitě a kterou diskrétně. Ve většině rovnic matematické fyziky, ve kterých je jednou z nezávisle proměnných originálu čas, je výhodné čas zachovat jako spojitou proměnnou. Nemusí to být pravidlem a je nutné pro každý složitější případ se rozhodnout individuálně na základě rozboru tvaru počátečních a okrajových podmínek.

Kdybychom pro rovnici (2.1) zvolili x za spojitě proměnnou a čas jako diskrétně proměnnou, dostali bychom soustavu diferenciálních rovnic druhého řádu, ve které je známa okrajová podmínka pro $u(0,t) = u_1(t)$. Pro rovnici druhého řádu však potřebujeme znát ještě počáteční podmínky u derivací, které bychom museli obtížně hledat zkusmo tak, aby byla splněna druhá okrajová podmínka tj. pro $u(L,t) = u_2(t)$.

2.1.2 Poznámka k okrajovým problémům

Řešením parciálních diferenciálních rovnic je:

$$u = u(x, y, z, t)$$

v definičním oboru Q. Definiční obor Q budeme uvažovat ve formě kartézského součinu $Q = O \times I$. O je podprostor třírozměrného euklidovského prostoru E_3. I je interval, v němž se pohybují hodnoty argumentu t. Argumenty x, y, z mají většinou význam prostorových souřadnic v E_3, argument t vyjadřuje zpravidla čas. Řešení musí uvnitř ohraničeného prostoru O splňovat danou parciální diferenciální rovnici. Na hranici H podprostoru O musí řešení splňovat předepsané okrajové podmínky. V technické praxi se setkáme často se třemi základními typy okrajových podmínek.

1. Okrajová podmínka prvního druhu: na hranici H podprostoru O jsou předepsány hodnoty řešení převážně jako funkce času:

$$u\Big|_H = f_1(t)$$

2. Okrajová podmínka druhého druhu: na hranici H podprostoru O jsou předepsány hodnoty derivace řešení podle vnější normály u na hranici H jako funkce času:

$$\frac{\partial u}{\partial n}\Big|_H = f_2(t)$$

3. Okrajová podmínka třetího druhu: v bodech hranice H podprostoru O je předepsána lineární kombinace okrajových podmínek jako funkce času:

$$\alpha u\Big|_H + \beta \frac{\partial u}{\partial n}\Big|_H = f_3(t), \qquad (\alpha, \beta \in \Re)$$

V technické praxi se mohou vyskytnout i jiné typy okrajových podmínek.

2.1.3 Vedení tepla v tenké tyči

Máme řešit časový průběh teploty v tenké isolované tyči o délce 4 cm s teplotním koeficientem $a = 1$, jestliže rozložení teploty po délce tyče je popsáno rovnicí $u(x, 0) = -\frac{1}{4}x^2 + x$ a konce tyče jsou udržovány na nulové teplotě.

Vedení tepla v tenké tyči můžeme popsat parabolickou difuzní rovnicí (2.1).

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2} \tag{2.14}$$

Rozdělíme tyč na osm částí, tj. devět uzlových bodů u_0, u_1, u_2, $\cdots$, u_8 a v těchto bodech budeme řešit průběh teploty v závisloti na čase. Druhou derivaci proměnné podle x nahradíme přibližnými diferenčními vzorci:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{(\Delta x)^2}$$

Danou diferenciální rovnici můžeme potom vyjádřit ve tvaru:

$$u_{k,i-1} - 2u_{k,i} + u_{k,i+1} = w \frac{du_k}{dt} \tag{2.15}$$

kde $w = \frac{h^2}{a}$; je-li $h = 0,5$, $a = 1$, bude $w = 0,25$. Provedeme-li transformaci času $T = 0,25$, pak můžeme psát $p = 4P$ a rovnici (2.15):

$$u_{k,i-1} - 2u_{k,i} + u_{k,i+1} = Pu_k \tag{2.16}$$

Počáteční podmínka: $u(x,0) = -\frac{1}{4}x^2 + x$

$x/cm/$	0	0.5	1	1,5	2	2.5	3	3.5	4
$u(x,0)$	0	0.4372	0.75	0.9375	1	0.9375	0.75	0.4375	0

Okrajové podmínky:

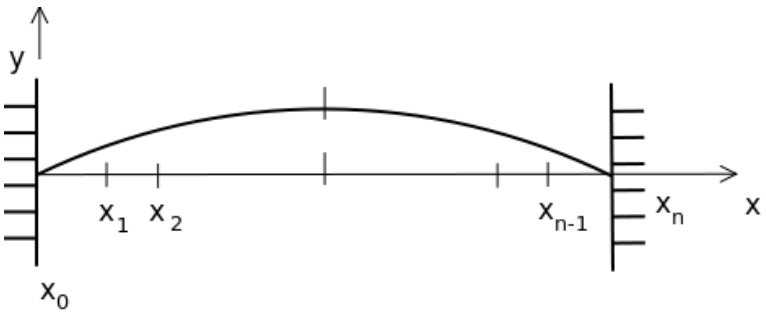
$$u_0(t) = u_0(t) = u(0,t) = 0 \tag{2.17}$$
$$u_8(t) = u_8(t) = u(4,t) = 0 \tag{2.18}$$

Rovnici napíšeme ve tvaru vhodném pro řešení na analogovém počítači.

$$\begin{array}{l|l}
1 & u_1' = \quad (-u_0 + 2u_1 - u_2) \\
2 & u_2' = \qquad (u_1 - 2u_2 + u_3) \\
3 & u_3' = \qquad\quad (-u_2 + 2u_3 - u_4) \\
4 & u_4' = \qquad\qquad (u_3 - 2u_4 + u_5) \\
5 & u_5' = \qquad\qquad\quad (-u_4 + 2u_5 - u_6) \\
6 & u_6' = \qquad\qquad\qquad (u_5 - 2u_6 + u_7) \\
7 & u_7' = \qquad\qquad\qquad\quad (-u_6 + 2u_7 - u_8)
\end{array}$$

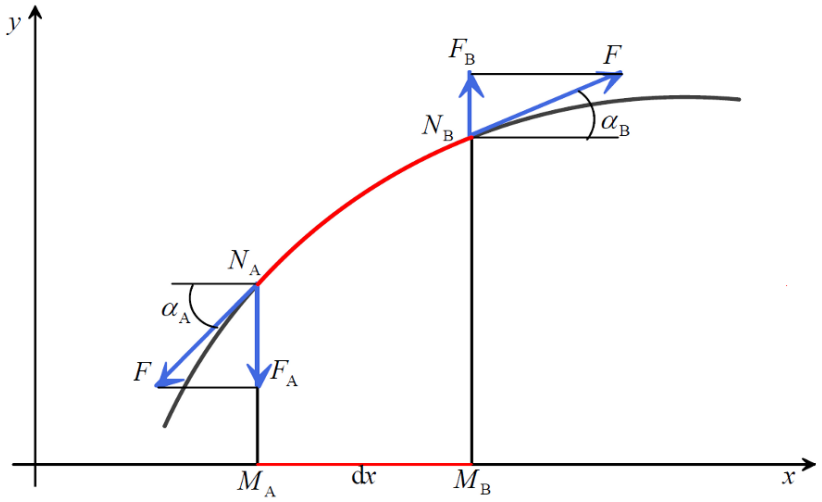
2.1.4 Kmitající struna - vlnová rovnice

V této části se zaměříme na řešení parciální diferenciální rovnice kmitající struny (vlnová rovnice). Vyšetřujeme strunu, která je vychýlena z rovnovážné polohy silou F a začne kmitat.



Obrázek 2.1: Kmitající struna.

Předpokládáme, že kmity jsou tak 'malé', že můžeme napětí ve všech bodech struny považovat za konstantní (F) a je možno zanedbat rozdíl mezi hodnotami $\tan(\alpha)$ a $\sin(\alpha)$. Struna je tedy napínána napětím F ve směru tečny v daném bodě. Na obrázku 2.2 vidíme znázorněnou sílu napětí F v bodech N_A a N_B struny, kam se přemístily body M_A a M_B po vychýlení z rovnovážné polohy. Úsek mezi body N_A a N_B má délku dx a hmotnost $dm = \rho dx$.



Obrázek 2.2: Kmitající struna.

Na body působí síly F_A, F_B, což jsou průměty sil, kterými působí zbytek struny na body N_A, N_B. Tyto síly jsou dány vztahy:

$$F_A = F\sin(\alpha_A), \qquad F_B = F\sin(\alpha_B) \tag{2.19}$$

Na celý úsek dx pak působí síla, jež je dána rozdílem:

$$dF = F_B - F_A = F\sin(\alpha_A) - F\sin(\alpha_B) \tag{2.20}$$

Díky možnosti zanedbání rozdílu mezi hodnotami $\tan(\alpha)$ a $\sin(\alpha)$ nahradíme $\sin(\alpha)$ směrnicí:

$$\tan(\alpha) = \frac{\partial y}{\partial x}$$

a dostáváme rovnici (2.20) ve tvaru:

$$dF = F\left[\frac{\partial u(x+dx,t)}{\partial x} - \frac{\partial u(x,t)}{\partial x}\right] \tag{2.21}$$

Dle druhého Newtonova zákona je výslednice sil působící na úsek dx rovna součinu hmotnosti a zrychlení. Rychlost struny je dána $v = \frac{\partial y}{\partial t}$, pak zrychlení $a = \frac{\partial v}{\partial t} = \frac{\partial^2 y}{\partial t^2}$. Výslednice sil je tedy:

$$dF = \rho dx \frac{\partial^2 y}{\partial t^2} \tag{2.22}$$

Rovnice 2.21 a 2.22 se rovnají:

$$\rho dx \frac{\partial^2 y}{\partial t^2} = F\left[\frac{\partial u(x+dx,t)}{\partial x} - \frac{\partial u(x,t)}{\partial x}\right] \tag{2.23}$$

a převedeme-li 2.23 na diferenciál dostáváme:

$$\frac{\partial y(x+dx,t)}{\partial x} - \frac{\partial y(x,t)}{\partial x} = \frac{\partial}{\partial x}\left(\frac{\partial y}{\partial x}\right)dx \tag{2.24}$$

Obě strany rovnice vydělíme výrazem ρdx:

$$\frac{\partial^2 y}{\partial t^2} = \frac{F}{\rho}\frac{\partial^2 y}{\partial x^2} \tag{2.25}$$

Označíme-li $a = \frac{F}{\rho}$ dostane tvar:

$$\frac{\partial^2 y}{\partial t^2} = a\frac{\partial^2 y}{\partial x^2} \tag{2.26}$$

okrajové podmínky:

$$y(0,t) = y(1,t) = 0$$

a platí:

$$\frac{d^2 y_i}{dt^2} = \frac{a_i}{\Delta x^2}(y_{i+1} - 2y_i + y_{i-1}) \tag{2.27}$$

Rovnice bude mít ve všech bodech stejný tvar až na body x_1 a x_{n-1}, kde se situace zjednoduší nulovými okrajovými podmínkami. Uvažujme oba konce struny upevněné, tzn. y_0 a y_n budou nulové ve všech okamžicích. To znamená, že budou platit vztahy pro bod 1 na levém konci:

$$\frac{a_1}{(\Delta x)^2}(y_2 - 2y_1) = \frac{d^2 y_1}{dt^2} \tag{2.28}$$

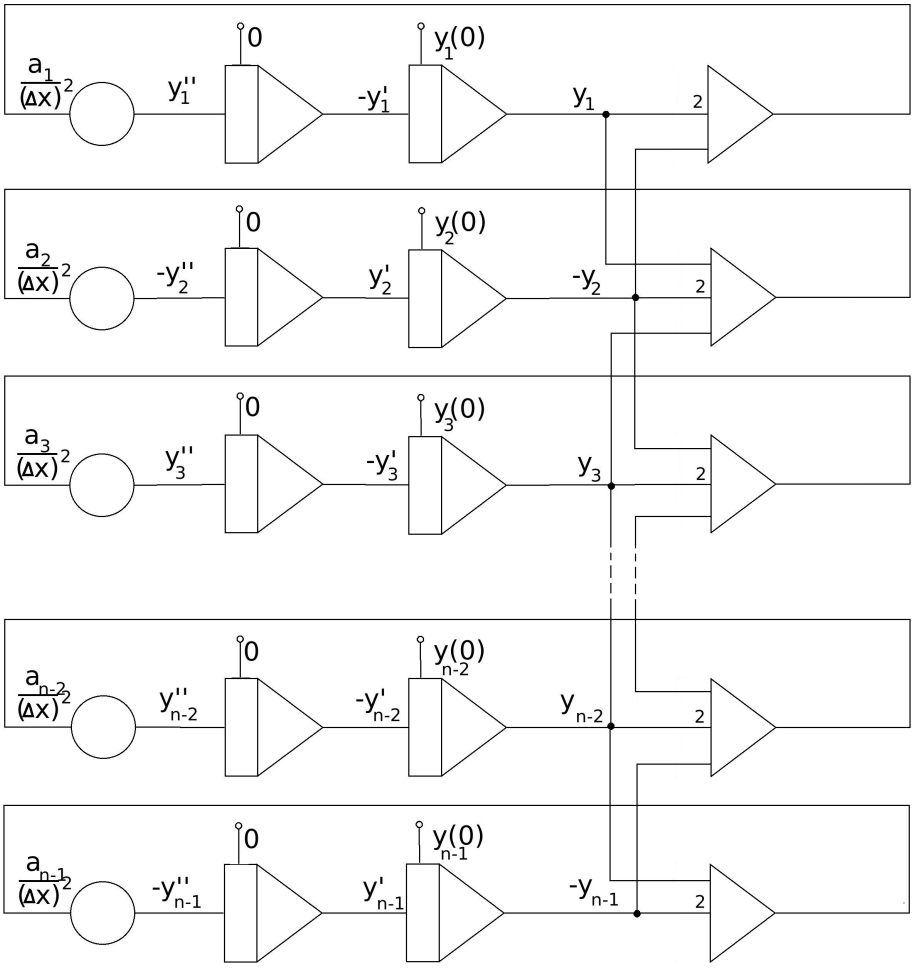
a pro předposlední bod $n-1$ na pravém konci:

$$\frac{a_{n-1}}{(\Delta x)^2}(y_{n-2} - 2y_{n-1}) = \frac{d^2 y_{n-1}}{dt^2} \tag{2.29}$$

Byla-li struna v počátku vychýlena souměrně a má-li souměrné vlastnosti (tj. a_i po délce konstantní) můžeme si zjednodušit řešení řešením pouze jedné poloviny.

Programové schéma rovnic 2.27 - 2.29 je na obrázku 2.3.

Jak vidíme z obrázku 2.3 při výpočtech se uplatní jednovstupý integrátor. Z obrázku také vidíme, že pro řešení tohoto konkrétního problému je zapotřebí $n-1$ integrátorů (pro $n-1$ rovnic). Tyto integrátory vždy nezávisle počítají nad příslušnými vstupními hodnotami, neboť odpovídající výrazy ve stejných řádech integrační metody nejsou na sobě závislé. Každý integrátor pracuje s rozdílnými daty, avšak používá stejný postup. Integrátory tak mohou být řízeny z jedné řídící jednotky, která zařídí výpočet všech hodnot k-tého kroku a poté se může přejít k výpočtu kroku $k+1$. Díky tomuto můžeme řešení velmi efektivně paralelizovat a dosáhnout tak výrazného zrychlení. V další kapitole se zaměříme na návrh jednovstupého integrátoru s počáteční podmínkou a navrhneme jej s ohledem na požadavek komunikace s případnými dalšími integrátory a centrálním řízením.



Obrázek 2.3: Programové schéma rovnic 2.27 - 2.29.

Kapitola 3

Numerický integrátor

V předchozí kapitole jsme naznačili řešení parciálních diferenciálních rovnic a to zjednodušením na soustavu obyčejných diferenciálních rovnic. V této kapitole navrhneme numerický integrátor, který se využije při řešení soustavy rovnic uvedených na obrázku 2.3. Návrh bude závislý na jednokrokové numerické integrační metodě využívající Taylorovu řadu [9]. Přesněji v prvotním návrhu použijeme Eulerovu metodu, která uvažuje pouze první dva členy Taylorova rozvoje. Zjednodušeně řečeno bude muset náš integrátor řešit následující rovnici:

$$y' = y \qquad y(0) = y_0 \tag{3.1}$$

Rovnice Eulerovy metody:

$$y_{i+1} = y_i + hy'_i \tag{3.2}$$

Po dosazení rovnice 3.1 do 3.2 dostáváme:

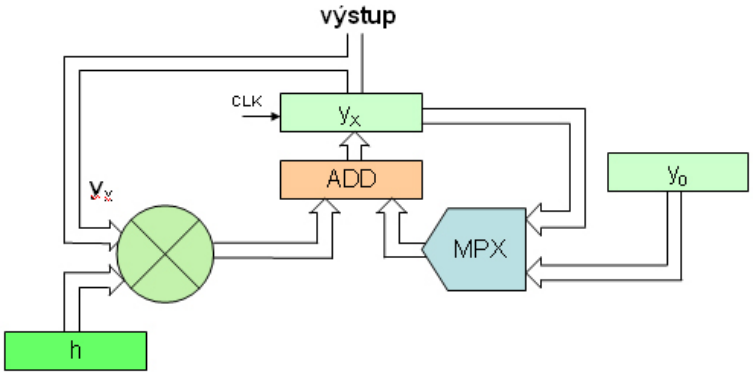
$$y_{i+1} = y_i + hy_i \tag{3.3}$$

Jak vidíme z rovnice 3.2, náš integrátor bude muset implementovat následující výpočetní operace:

- sčítání,
- odečítání,
- násobení.

Obrázek 3.1 nám schématicky řeší výpočet rovnice 3.3. V registrech se nachází počáteční hodnota y_0 a hodnota kroku h. Během prvního výpočtu je multiplexor nastaven na propagaci hodnoty počáteční podmínky y_0. Tím dojde k vyčíslení rovnice $y_1 = y_0 + hy_0$. V dalších krocích je již nastaven na přenos aktuální hodnoty y_x. Aby nedocházelo k nežádoucímu chování, musí být všechny prvky synchronizovány stejným hodinovým signálem CLK. Tento obrázek nám tedy schématicky ukazuje řešení Eulerovy metody obvodovou realizací.

Náš návrh bude vycházet z obrázku 3.1. Budeme pracovat ve dvojkové soustavě a před samotným návrhem si zrekapitulujeme uložení čísel v pevné a plovoucí řádové čárce včetně potřebných operací s nimi.



Obrázek 3.1: Obvodové řešení rovnice (3.3).

3.1 Čísla s pevnou řádovou čárkou

Při zobrazení čísla v pevné řádové čárce je číslo zakódováno do n bitů tak, že prvních m bitů odpovídá celé části a zbylých d bitů odpovídá zlomkové části [20]. Takové kódování se pak označuje jako kódování $m.d$. Pro reprezentaci záporných čísel lze použít přímý kód, doplňkový kód, inverzní kód a aditivní kód [5]. První možný způsob vyčleňuje MSB (most significant bit) znaménku celého čísla. Hodnota 0 MSB označuje kladné číslo a MSB rovna hodnotě 1 značí číslo záporné. Vzhledem k principu, obsahuje toto kódování dvě nuly(kladnou a zápornou) a je nutno vždy testovat znaménkový bit a podle výsledku provést sčítání nebo odčítání. Proto byl později zaveden doplňkový kód pro záznam záporných čísel. Při kódování v doplňkovém kódu je provedena binární negace všech bitů původního čísla zvětšena o jedničku. Výhodou kódování je, že pro operace sčítání a odečítání lze použít stejnou sčítačku.

Inverzní kód reprezentuje záporná čísla binární negací. Aditivní kód přičitá k číslům předem známou konstantu c. Nula je tedy vyjádřena právě touto konstatnou a záporná čísla padají do intervalu $< 0, c)$ a kladná čísla se nacházejí v intervalu $(c, 2 * c)$. Pokud nebude explicitně uvedeno jinak, budeme pracovat v doplňkovém kódu.

3.1.1 Součet

Podmínkou pro uskutečnění součtu je shodný počet bitů operandů vstupujících do operace. Pokud není podmínka splněna je nutno před samotným sečtením provést korekci bitovým posuvem. Je-li splněna podmínka, je součet dán vztahem:

$$A * 2^b + B * 2^b = (A + B) * 2^b$$

Základním stavebním prvkem pro sečítání je úplná jednobitová sčítačka, jejíž logika je uvedna v tabulce 3.1. Sčítačka provede sečtení obou vstupů (X_i, Y_i) a přenosu C_{i-1}. Výsledek je pak na výstupu v Z_i s případným přenosem C_i. Propojujeme vždy potřebný počet těchto sčítaček, kolika bitové operandy vstupují do operace sečítání. Na vstupy přivedeme příslušné bity operandů a v čase $n * 2T$ dostáváme přenos n-té sčítačky a n-tý bit výsledku.

C_{i-1}	Y_i	X_i	Z_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabulka 3.1: Úplná jednobitová sčítačka.

3.1.2 Rozdíl

Operaci odečtení lze převést na sčítání převodem menšitele za pomoci doplňkového kódu (viz kapitola 3.1). Obvod sčítačky se tak doplní o logiku převodu čísla do doplňkového kódu jednoho z operandů.

3.1.3 Násobení

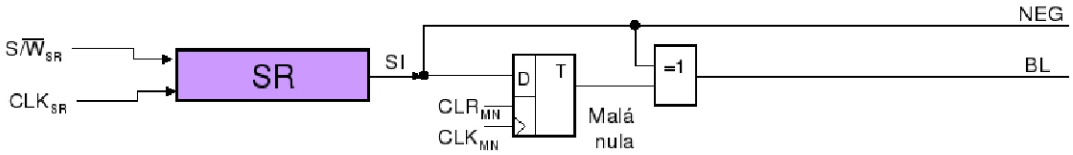
Klasické postupy pro sekvenční násobení jsou nepraktické (viz [4]). Pro násobení čísel se znaménkem existuje celá řada algoritmů. Nejrozšířenějším je Boothův algoritmus násobení [8]. Algoritmus je založen na dílčích součtech, rozdílech a posuvech. Během výpočtu dochází k porovnávání jednotlivých sousedních bitů násobitele a toto porovnávání určuje prováděnou operaci dle tabulky 3.2.

log. hodnota bitů násobitele		
b_i bit	b_{i-1} bit	odpovídající akce
0	0	přičtení nul k mezivýsledku
0	1	přičtení násobence k mezivýsledku
1	0	odečtení násobence od mezivýsledku
1	1	přičtení nul k mezivýsledku

Tabulka 3.2: Princip Boothova algoritmu.

Násobitel je uložen v posuvném registru a je postupně posouván. Aktuální nejméně významový bit je porovnáván s předchozí hodnotou, která je uložena v D klopném obvodu (první porovnání MSB bitu násobitele je provedenou vždy s hodnotou 0). Díky tomu se k testu sousedních bitů používá jeden vodič. Výsledek dle odpovídající akce je přičten k akumulátoru, kde se po skončení algoritmu (porovnání všech bitů násobitele) nachází i výsledek. Dle tabulky 3.2 tedy musíme zajistit jak přičítání samotného násobence, tak i přičtení jeho dvojkového doplňku. Samotný násobenec je uložen v registru SIR (RN). Odtud je jeho hodnota propagována nezměněna přes registr $BNEG$ (signál $NEG = 0$) nebo jsou jeho bity invertovány ($NEG = 1$). Dle signálu BL je hodnota $BNEG$ dále propagována ($BL = 1$) nebo jsou všechny bity vynulovány ($BL = 0$).

Možná obvodová realizace obvodu testování sousedních bitů je uvedena na obrázku 3.2.



Obrázek 3.2: Posuvný registr násobitele a logika nastavující signály NEG a BL.

Pro úplnost ještě uveďme v tabulce 3.3 závislost signálu BL, NEG na hodnotách sousedních bitů.

b_{last}	bit v D klopném obvodu	BL	NEG
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Tabulka 3.3: Závislost signálů BL a NEG na srovnávaných sousedních bitech.

Na obrázku 3.3 vidíme obvodovou realizaci násobičky, jež využívá již zmíněné prvky a realizuje násobení pomocí Boothova algoritmu.

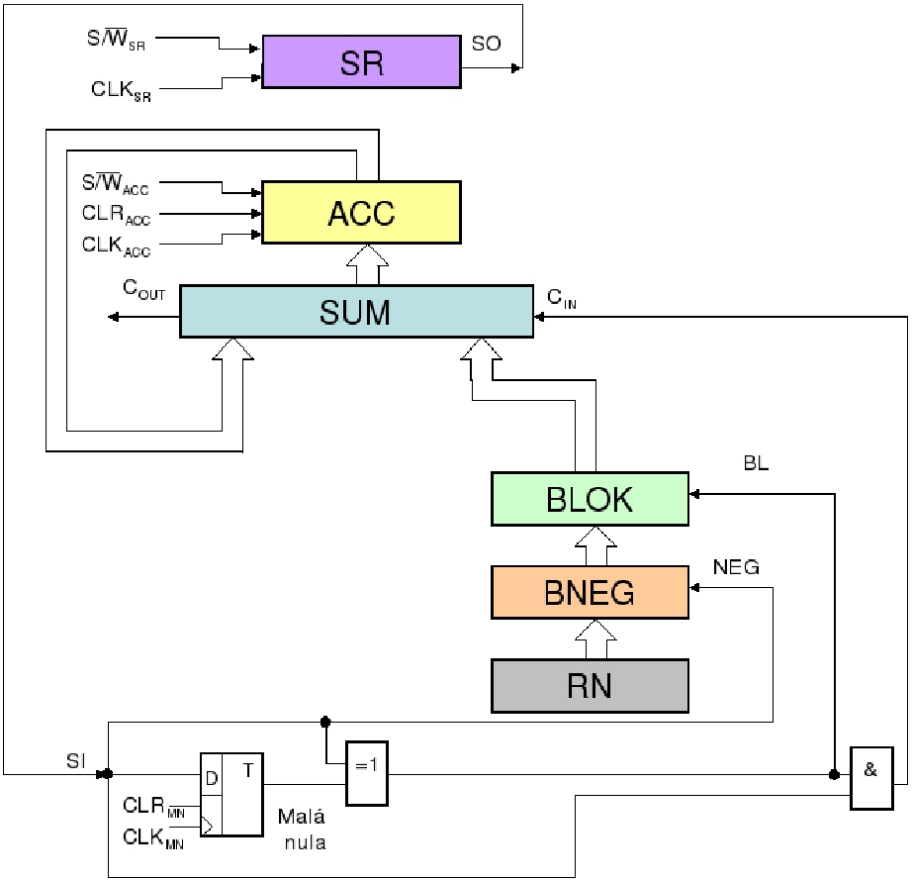
3.2 Čísla s plovoucí řádovou čárkou

Formát čísel v pohyblivé řádové čárce (FP) je popsán v normě IEEE 754 [7]. Tato norma také obsahuje pravidla pro operace se dvěma nejpoužívanějšími formáty pro uložení čísel s plouvocí řádovou čárkou. Jedná se o formát single normy IEEE 754 a double formát. Oba formáty se liší v počtu bitů připadajících na mantisu a exponent. Formát double používá jedenáct bitů na exponent a padesát dva bitů tvoří mantisu. Formát single si popíšeme blíže.

Single formát je tvořen znaménkovým bitem (s), za kterým následuje osm bitů pro uložení posunutého exponentu (exp). Zbylých dvacet tři bitů slouží pro uložení mantisy (m). Vzorec pro vyjádření dekadické hodnoty je pak:

$$X_{single} = (-1)^s * 2^{exp-127} * (1.m) \tag{3.4}$$

Jak vidíme ze vzorce 3.4 je-li znaménkový bit roven jedné, jedná se o číslo záporné, v opačném případě je uložena hodnota kladná. Exponent je uložen v tzv. posunuté formě, kdy posunutí je dáno vzorcem $bias = 2^{b-1} - 1$, kde b je počet bitů vyhrazených pro mantisu. Tedy v případě single formátu je posunutí 127 a u double formátu 2047. Obě krajní hodnoty exponentu (všechny bity buď jedničkové či nulové) jsou použity pro vyjádření speciálních hodnot. Všechny speciální hodnoty včetně jejich reprezentace lze najít v normě IEEE 754 [7]. Jak vidíme z rovnice 3.4, číslo je uloženo ve tvaru, kdy se první jednička mantisy neukládá. Postupu převodu čísla na tento tvar se říká normalizace. Pokud hodnotu nelze normalizovat, je exponent nastaven na nejnižší hodnotu a nejvyšší (explicitně neukládaný) bit je nula.



Obrázek 3.3: Násobička.

3.2.1 Součet

Operace v aritmetice pohyblivé řádové čárky se rozpadají na operace s exponenty a mantisami [21]. Předpokládejme, že do operace nevstupují speciální hodnoty. Nejprve je nutné porovnat exponenty. Nejsou-li shodné, provede se posun mantisy menšího čísla doprava o číselný rozdíl exponentů. Na základě znaménkových bitů je vyčíslen buď rozdíl, nebo součet mantis. Pokud se hodnota nevejde do počtu bitů určených pro mantisu, posunou se bity mantisy doprava a exponent se inkrementuje. K tomu, aby se mohlo toto přetečení detekovat, je nutné provádět součet na sčítačce o jeden bit vyšší, než je bitová délka mantisy. Dále se provede kontrola normalizace a je vyjádřeno znaménko výsledku.

Dosáhne-li exponent během výpočtu své maximální hodnoty, dojde k přetečení výsledku. Tento stav nelze obejít bez zvýšení rozsahu, tj. volbou jiného formátu. Při této skutečnosti je výsledek operace sčítání kladné nekonečno. V případě odečítání pak nekonečno záporné. Dosáhne-li hodnota exponentu své minimální hodnoty, dochází k podtečení. Jako výsledek je vrácena, dle vypočítaného znaménka výsledku, kladná či záporná nula.

3.2.2 Násobení

Mějme dány dva operandy:

$$X = (-1)^{s_X} * 2^{exp_X} * m_X, \qquad Y = (-1)^{s_Y} * 2^{exp_Y} * m_Y$$

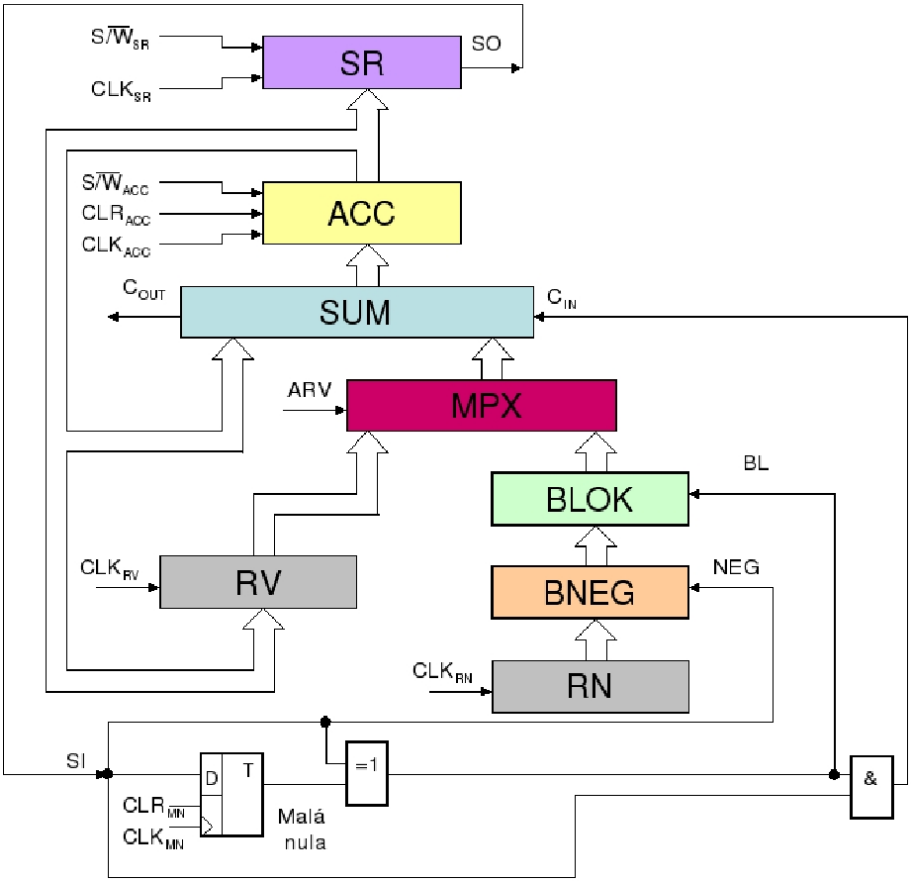
Výsledek násobení dostaneme vynásobením mantis a sečtením exponentů. Výsledný znaménkový bit je dán operací nonekvivalence nad nejvýznamnějšími bity operandů. Tedy:

$$X * Y = (-1)^{s_X} * (-1)^{s_Y} * 2^{exp_X + exp_Y} * m_X * m_Y$$

Stejně jako u součtu je nutno s výsledkem provést normalizaci a případné zaokrouhlení.

Vzhledem ke značné rozdílnosti práce s čísly uloženými v pevné a plovoucí řádové čárce, uvedeme v této práci postupně návrh numerického integrátoru v pevné řádové čárce a na základě tohoto návrhu ukážeme i realizaci v plovoucí řádové čárce.

3.3 Numerický integrátor pracující s čísly v pevné řádové čárce



Obrázek 3.4: Numerický integrátor.

Základ integrátoru tvoří obvodová realizace násobení pomocí Booth algoritmu (viz 3.1.3). Najdeme zde tedy registr s paralelním vstupem a paralelním výstupem [10] RN, kde je uložen násobenec. Tento registr je tvořen D klopnými obvody viz [4]. Blok řízené negace BNEG, jež v závislosti na signálu NEG provádí negaci vstupních dat ($NEG = 1$), nebo tyto data nemění ($NEG = 0$). BNEG lze realizovat pomocí registru, jež má svůj přímý i negovaný vstup připojen na multiplexor a činnost multiplexoru je řízena právě

signálem NEG. Blokovací obvod BNEG, sčítačka SUM a akumulátor s aritmetickým posuvem ACC, kam je ukládána průběžná hodnota a po skončení algoritmu Booth násobení se zde nachází výsledek. Sčítačka SUM krom vstupu pro dva operandy obsahuje jednobitový vstup C_{in}. Tento bit je zde pro realizování doplňkového kódu (dvojkový doplněk pomoci obvodu BNEG plus přičtení 1 v podobě nastaveného bitu $C_{in} = 1$). Dále je zde registr pro uložení hodnoty násobitele SR. Tento registr musí být posuvný a je navržen tak, aby prováděl logický posuv vpravo. Při logickém posuvu se chybějící hodnota doplňuje hodnotou 0, zatímco u aritmetického posuvu se postupně kopíruje hodnota před posuvem. Pro úplnou realizaci Booth algoritmu nechybí v integrátoru logika nastavování signálu BL, NEG a C_{in} za pomoci testování sousedních bitů násobitele.

Krom těchto prvků obsahuje numerický integrátor registr pro uchování celkových výsledků RV a multiplexor MPX. Výsledné schéma je uvedeno na obrázku 3.4.

3.3.1 Činnost numerického integrátoru

Před každým výpočtem jsou registry vynulovány včetně klopného obvodu. Vynulováním klopného obvodu je zaručeno prvotní porovnání posledního bitu násobitele s nulou. Do registru SR a RV je nahrán násobenec a do registru RN násobitel. Multiplexor je nastaven tak, aby propagoval hodnotu z bloku řízene negace a blokování. V takto nastaveném numerickém integrátoru je spuštěn algoritmus násobení dle 3.1.3. Poté je k výsledku násobení přičtena hodnota registru RV. Celkové řízení je realizováno řadičem. Rozlišujeme mnoho druhů řadičů [4], přičemž k řízení našeho integrátoru využijeme mikroprogramový řadič (obr. 3.5). Ten je tvořen překrývacím registrem a pamětí, ve které je uložen mikroprogram. Překrývací registr má paralelní vstup a výstup a je řízen signály CLR_{PR} (nulování všech klopných obvodů) a CLK_{PR}, jehož nástupná hrana provede zápis dat do registru.

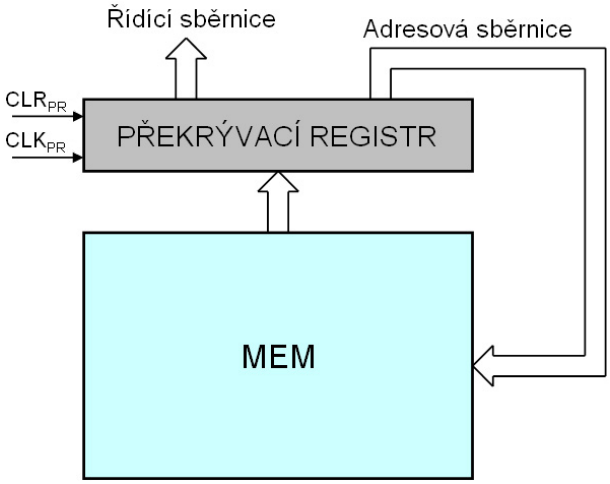
Data vstupující do překrývajícího registru obsahují dvě části. V první se nacházejí informace ovládající řídící signály mikroprocesoru a druhá část obsahuje adresu následující instrukce. Tabulka 3.4 obsahuje mikroprogram začínající na adrese 0000. Od této adresy až do adresy 1000 se nacházejí instrukce řídící operaci násobení (viz 3.1.3). Řídící signály násobení jsou CLR_{ACC}, CLR_{MN}, $S/\overline{W}_{ACC}$, $S/\overline{W}_{SR}$, CLK_{ACC}, CLK_{SR}. Signál CLK_{MN} v programu chybí, neboť je shodný se signálem CLK_{SR} z důvodu nutné synchronizace posuvu v registru SR s nahráním nejnižšího významového bitu do D klopného obvodu. Řídící signál ARV řídící MPX zajišťuje přepínání datových vstupů do sčítačky. CLK_{RV} slouží k uložení hodnoty do registru RV. Před spuštěním samotného programu je vynulován překrývací registr signálem CLR_{PR}. S nástupnou hranou signálu CLK_{PR} dojde k načtení instrukce a se sestupnou hranou je tato instrukce vybavena z paměti (nachází se na výstupu paměti) tak, aby se znovu mohla s nástupnou hranou signálu CLK_{PR} načíst. Tím je zajištěno generování nové hodnoty bitů řídící sběrnice (CLR_{ACC}, CLR_{MN}, $S/\overline{W}_{ACC}$, $S/\overline{W}_{SR}$, CLK_{ACC}, CLK_{SR}, CLK_{RV}, ARV), které požadovaným způsobem řídí numerický integrátor.

3.4 Numerický integrátor pracující s čísly v plovoucí řádové čárce

Při návrhu budeme vycházet z již uvedených poznatků. Především budeme vycházet z kapitoly pojednávající o číslech reprezentovaných ve formátu plovoucí řádové čárky (viz 3.2), kde jsme si ukázali, že operace v aritmetice pohyblivé řádové čárky se rozpadají na operace

CLR_{ACC}	CLR_{MN}	$S/\overline{W}_{ACC}$	$S/\overline{W}_{SR}$	CLK_{ACC}	CLK_{SR}	CLK_{RV}	ARV	INSTR
`-- 0000 -- vynulování ACC a D klopného obvodu v obvodu Malé nuly:`								
1	1	0	1	0	0	0	0	0001
-- 0001 -- `ukončení nulování:`								
0	0	0	1	0	0	0	0	0010
-- 0010 -- `ACC do režimu zápisu:`								
0	0	0	1	0	0	0	0	0011
-- 0011 -- `zápis mezisoučtu do ACC:`								
0	0	0	1	1	0	0	0	0100
-- 0100 -- `konec zápisu do ACC:`								
0	0	0	1	0	0	0	0	0101
-- 0101 -- `nastavení ACC na posuv:`								
0	0	1	1	0	0	0	0	0110
-- 0110 -- `posuv ACC i SR vpravo:`								
0	0	1	1	1	1	0	0	0111
-- 0111 -- `konec hodin posuvu:`								
0	0	1	1	0	0	0	0	1000
if (posunut celý registr násobitele vpravo) instrukce=1001 else instrukce=0011								
-- 1000 -- `ukončení nulování:`								
0	0	0	1	0	0	0	0	1001
-- 1001 -- `přepnutí MPX:`								
0	0	0	1	0	0	0	1	1010
-- 1010 -- `zápis výsledku do ACC:`								
0	0	0	1	1	0	0	1	1011
-- 1011 -- `zápis výsledku do RV:`								
0	0	0	1	0	0	1	1	1100

Tabulka 3.4: Mikroprogram numerického integrátoru.



Obrázek 3.5: Mikroprogramový řadič.

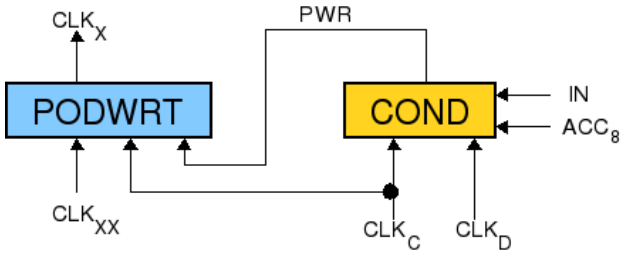
s mantisami a exponenty. Proto bude nutné rozšířit návrh o některé další prvky, které si uvedeme na začátku této kapitoly.

Zaměříme se na dva možné přístupy k řešení numerické integrace a to paralelní zpracování mantisy a exponentu a zpracování postupné. Paralelní řešení bude vyžadovat návrh dvou dílčích jednotek zatím co sériový přístup bude proveden v jedné společné jednotce.

3.4.1 Dílčí prvky numerického integrátoru

Obvod podmíněného zápisu – PC

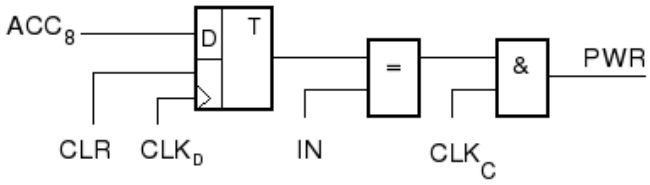
Chceme-li sečíst dva operandy v aritmetice plovoucí řádové čárky, musí mít stejné exponenty. Pokud tomu tak není, musíme jeden operand posunout do prava o tolik pozic, kolik činí rozdíl jejich operandů. Při výpočtu rozdílu můžeme dostat nezáporný rozdíl, je-li exponent prvního operandu vyšší nebo roven druhému, nebo záporný rozdíl, je-li exponent prvního operandu menší. V prvním případě provedeme posuv nad mantisou druhého operandu, respektive při záporném rozdílu budeme posouvat mantisu prvního operandu. K řízení posunutí nám pomůže právě obvod podmíněného zápisu.



Obrázek 3.6: Obvod podmíněného zápisu.

Na obrázku 3.6 je uvedeno schéma obvodu podmíněného zápisu, který je tvořen dvěma částmi. Obvod podmínky COND a obvod zápisu PODWRT. Obvod podmínky COND na

obrázku 3.7 slouží k výrobě signálu pro podmíněný zápis do registru. Tento signál je dán srovnáním znaménkového bitu střadače ACC s řídícím signálem IN (předpokládáme uložení hodnoty na osmi bitech - znaménkový bit je ACC_8). Aktivní signál PWR je pak tou podmínkou, která blokuje či povoluje hodinový signál zápisu do registru. Funkci obvodu pro jednotlivé kombinace signálu nám ukazuje tabulka 3.5. Spolu se signálem CLK_C je i signál PWR veden do obvodu zápisu PODWRT.



Obrázek 3.7: Detail obvodu COND.

Vstupy			Výstup
ACC_s	IN	CLK_C	PWR
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	1

Tabulka 3.5: Obvod podmínky COND.

Obvod podmíněného zápisu na základě vstupních signálů z obvodu podmínky COND (CLK_C, PWR) a řídícího signálu CLK_{XX} z řadiče vytváří hodinový signál CLK_X pro jednotlivé registry (X a XX je univerzální označení, ve schématu je použito konkrétní označení dle příslušného registru). Registry, u kterých uplatňujeme podmíněný zápis, nejsou ovládány řadičem přímo, ale hodinový signál prochází tímto obvodem podmíněného zápisu. Proto je nutné mít pro každý registr příslušný obvod PODWRT.

Vstupy			Výstup	
CLK_{XX}	PWR	CLK_C	CLK_X	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	0	c
1	0	0	1	a
1	0	1	0	d
1	1	0	0	
1	1	1	1	b

Tabulka 3.6: Závislost výstupu hodinového signálu pro daný registr na obvodu PODWRT a jeho vstupních signálech.

Na základě tabulky 3.6 můžeme rozlišit čtyři nejvýznamější stavy:

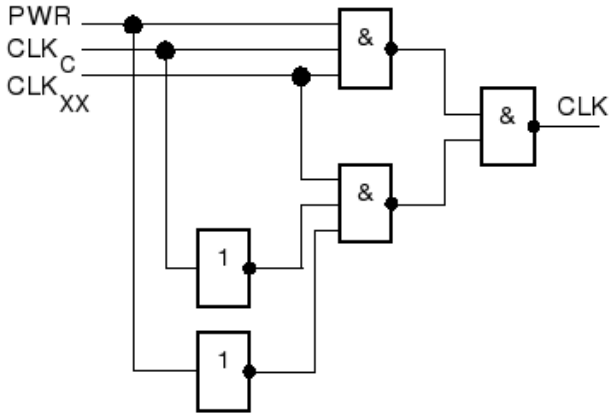
a) Je-li hodinový signál $CLK_{XX} = 1$ a současně signály PWR i $CLK_C = 0$, uskuteční se obvyklý (nepodmíněný) zápis do registru. Hodinový signál $CLK_C = 0$ označí, že obvod podmínky $COND$ není v činnosti.

b) Jsou-li signály CLK_{XX}, PWR i $CLK = 1$, uskuteční se podmíněný zápis, signál $CLK_C = 1$ značí, že obvod podmínky $COND$ je aktivní.

c) Je-li signál $CLK_{XX} = 0$ a signály PWR i $CLK_C = 1$ značí to, že obvod podmínky $COND$ je aktivní, ale signálem $CLK_{XX} = 0$ je blokován zápis řadičem do registru.

d) Je-li signál CLK_{XX} i $CLK_C = 1$ a signál $PWR = 0$ značí to, že se jedná o podmíněný zápis, který se však neuskuteční. Není splněna podmínka rovnosti signálu IN a znaménkového bitu střadače ACC ($PWR = 0$).

Možná realizace obvodu PODWRT tak, aby splňoval funkci tabulky 3.6 je uvedena na obrázku 3.8.

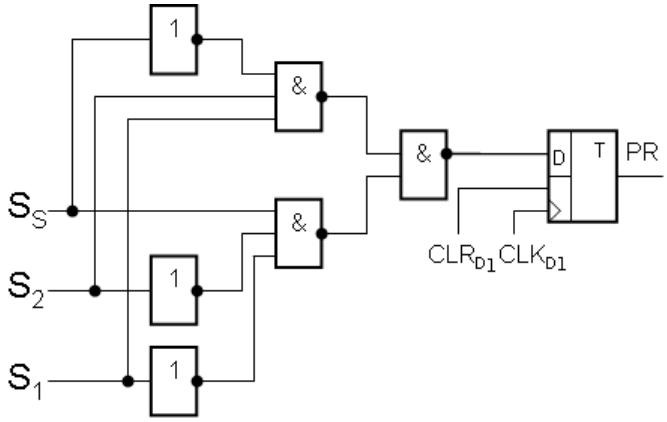


Obrázek 3.8: Detail obvodu PODWRT.

Obvod ošetření přetečení

Při sečítání operandů může dojít k přetečení. V aritmetice pohyblivé řádové čárky je přetečení ošetřeno posuvem mantisy vpravo a inkrementací exponentu. Přetečení lze detekovat obvodem na obrázku 3.9.

Jak vidíme ze zapojení 3.9, signál PR je nastavován znaménkovými bity operandů vstupujících do operace součtu (S_1, S_2) a znaménkovým bitem výsledku (S_S). Funkci obvodu přetečení pro jednotlivé kombinace znaménkových bitů ukazuje tabulka 3.7. Tabulka ukazuje, že signál PR je generován jen při kombinaci znaménkových bitů indikujících přetečení. Signál PR, který je veden do obvodu podmínky COND, je srovnáván s řídícím signálem IN. K ošetření přetečení dojde pouze tehdy, pokud jsou signály IN a PR rovny jedné. Ošetření lze blokovat hodinovým signálem CLK_C rovným nule v obvodě podmínky COND. Vzhledem k postupnému zpracování mantisy a exponentu je nutné si příznak přetečení pamatovat. K tomu slouží D klopný obvod, který je nutno po ošetření přetečení vynulovat.



Obrázek 3.9: Obvod ošetření přetečení.

Vstupy			Výstup
S_1	S_2	S_s	PR
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Tabulka 3.7: Obvod ošetření přetečení pro vstupující kombinace signálů.

Na počátku ošetření přetečení je mantisa operandu v ACC a exponent v registru RVE. S hodinovým signálem CLK_D se zapíše hodnota signálu PR do D klopného obvodu v obvodu COND a současně se nastaví signál $IN = 1$. Provede se posuv obsahu střadače ACC. Anuluje se D klopný obvod a signál IN se vrací do původního stavu (zrušení podmíněného zápisu). Následuje přesun mantisy do registru RV. Nyní se inkrementuje exponent. Opět se signálem CLK_D zapíše hodnota signálu PR do D klopného obvodu a nastaví se signál IN na hodnotu jedna. Poté je hodnota RVE sečtena se signálem přenosu C_{in} v ACC. Inkrementovaný exponent je uložen do RVE, nuluje se klopný obvod typu D a signál IN.

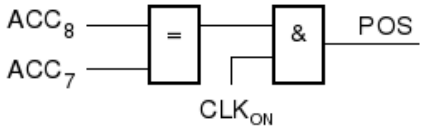
Obvod normalizace

Splňuje-li mantisa nerovnost:

$$z^{-1} <= |M| < 1 \tag{3.5}$$

kde z představuje základ použité soustavy, hovoříme o ní, jako o hodnotě normalizované. Obvod na obrázku 3.10 umožňuje detekovat normalizaci obvodovými prostředky. Vychází ze skutečnosti, že hodnoty prvních dvou nejvíce významových bitů jsou odlišné (uložení hodnot na osmi bitech). V bloku ekvivalence se tedy tyto hodnoty z registru SIR srovnají, čímž dostaneme požadovaný řídící signál pro posuvy vlevo, které se provádějí vždy v registru SIR. Aby nedošlo ke kolizním stavům (obvod normalizace pracuje při libovolném obsahu

registru SIR) je výstupní signál blokován signálem CLK_{ON}. Současně při posuvu je nutno dekrementovat exponent. Tato operace je zajištěna smyčkou ACC → sčítačka SUM s přenosem CI → ACC. Funkci obvodu normalizace nám pro jednotlivé kombinace vstupních signálů přibližuje tabulka 3.8. Pro správnou činnost obvodu je nutné, aby řídící signál IN byl roven jedné.



Obrázek 3.10: Obvod normalizace.

Vstupy		Výstup
SIR_8	SIR_7	POS
0	0	1
0	1	0
1	0	0
1	1	1

Tabulka 3.8: Obvod normalizace pro vstupující kombinace signálů.

Realizaci posuvu si ukážeme na následujících řídících příkazech za předpokladu umístění mantisy operandu v registru SIR a jeho exponentu ve střadači ACC. Výraz tvaru /XXX/ → YYY představuje přenos obsahu registru XXX do registru YYY.

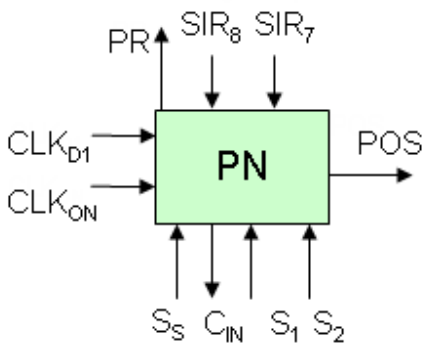
1. $IN = 1$, CLK_D, CLK_{ON}
2. `nastavení posuvu vlevo registru SIR`
3. CLK_{SIR}, CLK_{CI}
4. `/ACC/ +` $CI \rightarrow$ `ACC`

Signálem CLK_D je zapsána výstupní hodnota obvodu normalizace do klopného obvodu typu D v bloku podmíněného zápisu. Signál CLK_{ON} uvolní výstup obvodu normalizace (první řádek). Následuje nastavení posuvu registru SIR. Na třetím řádku se signálem CLK_{SIR} uskuteční posuv a současně je uvolněn přenos CI ve sčítačce SUM za pomoci signálu CLK_{CI}. Poslední řádek zajistí inkrementaci exponentu. Na závěr se vynuluje klopný obvod a signál IN. Znormalizovaná mantisa je uložena v registru SIR a její exponent ve střadači ACC.

Obvod normalizace budeme v našem schématu znázorňovat spolu s obvodem ošetření přetečení za pomocí obrázku 3.11.

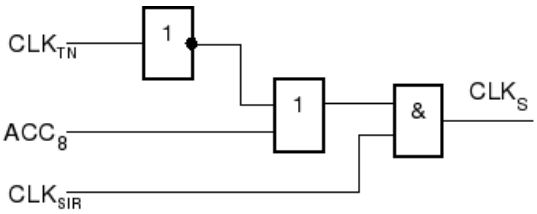
Obvod blokování hodin registru SIR

Při sečítání operandů s různými exponenty je třeba, aby jednotlivé řády mantis operandů odpovídaly. Někdy je tedy nutné jednu z mantis posunout o rozdíl exponentů. Budeme-li uvažovat více než jeden integrátor, nebude jistě ve všech integrátorech hodnota posunutí stejná. Proto je nutná přítomnost obvodu pro blokování hodin registru SRI v jednotlivých



Obrázek 3.11: Symbolická značka obvodu ošetření přetečení a normalizace.

integrátorech, nezávisle na centrálním řízení, aby nedošlo k chybnému posuvu. Obvod je uveden na obrázku 3.12.



Obrázek 3.12: Obvod blokování hodin registru SIR.

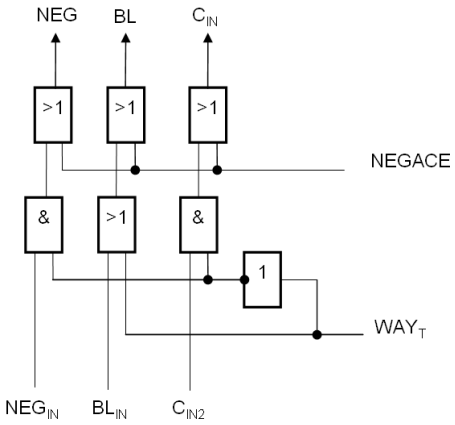
Výsledný signál CLK_S dostaneme na základě kombinace hodnot znaménkového bitu střadače ACC, hodinového signálu CLK_{TN}, který udává dobu aktivity daného obvodu. V tabulce 3.9 jsou pak zachyceny všechny možné kombinace vstupních signálů. Tabulku lze shrnout tak, že pokud je nutný posuv (ACC je stále záporný), nebo je povolena aktivita obvodu ($CLK_{TN} = 1$) je propagována hodnota CLK_{SIR}. Sloupec **a** nám označuje situaci, kdy už je upravena mantisa a je nutno blokovat hodiny registru.

						a		
CLK_{TN}	1	1	1	1	0	0	0	0
ACC_8	0	0	1	1	0	0	1	1
CLK_{SIR}	0	1	0	1	0	1	0	1
CLK_S	0	1	0	1	0	0	0	1

Tabulka 3.9: Obvod blokování hodin registru SIR.

Obvod negace a řízení signálů $BNEG$, $BLOK$ a C_{in}

Během výpočtu je potřeba provádět negaci i mimo násobení za pomocí Booth algoritmu. K ovládání signálů $BNEG$, $BLOK$ a C_{in} nám slouží obvod řízení uvedený na obrázku 3.13.



Obrázek 3.13: Obvod negace a řízení signálů $BNEG$, $BLOK$ a C_{in}.

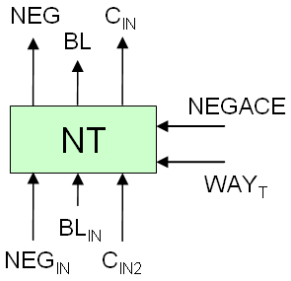
	WAY_T	$NEGACE$	NEG	BL	C_{in}
a	0	0	NEG_{in}	BL_{in}	C_{in2}
b	0	1	1	1	1
c	1	0	0	1	0
d	1	1	1	1	1

Tabulka 3.10: Obvod negace a řízení signálů.

Pokud jsou řídící signály obvodu WAY_T a $NEGACE$ v 0, jsou propagovány hodnoty tak, jak byly nastaveny srovnáním sousedních bitů operandu uloženého v registru SR (v tabulce 3.10 řádek a). Tento stav bude při výpočtu násobení. Bude-li potřeba vypnout řízení srovnáváním bitů, nastavíme signál WAY_T na logickou úroveň 1. Tím bude nastaven signál BL na 1, NEG a C_{in} na 0 (tabulka 3.10 řádek c). Posuvy mezi registrem SIR a sumátorem tak nebudou blokovány (obvod BLOK blokuje při logické úrovni 0 signálu BL). Budeme-li vyžadováno přičtení negované hodnoty k sumátoru, nastaví řadic signál $NEGACE$ na jedničku. Tím dojde k nastavení všech výstupních signálů na 1 (tabulka 3.10 řádek b). Bude tak provedeno negování hodnoty registru SIR, nebude provedeno blokování a k hodnotě sumátoru bude přičten přenos. Ze zapojení 3.13 vidíme, že signál $NEGACE$ má větší váhu a jsou-li oba řídící signály nastaveny na hodnotu jedna, je prováděno negování (řádek d). To nám umožňuje nastavit signál WAY_T po ukončení násobení na hodnotu jedna a potom jen nastavovat signál $NEGACE$ dle potřeby výpočtu. Symbolická značka obvodu je uvedena na obrázku 3.14.

3.4.2 Návrh a popis činnosti integrátoru zpracovávající mantisu a exponent v jedné společné jednotce

Nyní již známe význam jednotlivých prvků návrhu. Celkové zapojení numerického integrátoru pracujícího s čísly v plovoucí řádové čárce je uvedeno na obrázku 3.15. Integrátor provádí zpracování mantisy a exponentu operandů postupně. Posuvy mantisy při úpravách se zásadně provádí v registru SIR. Čítač posuvů je realizován smyčkou - střadač ACC



Obrázek 3.14: Symbolická značka obvodu negace a řízení signálů $BNEG$, $BLOK$ a C_{in}.

→ multiplexor MPX2 → sčítačka SUM s přenosem C_{in} → střadač ACC. Čítání probíhá tím způsobem, že do střadače se zapíše záporně vzatý rozdíl exponentů obou operandů a postupně se k obsahu ACC přičítá jednička. Ukončení posuvů nastane tehdy, když je znaménkový bit ACC roven nule.

Činnost numerického integrátoru

Algoritmus výpočtu si ukážeme na následující posloupnosti příkazů. Zápisy pro alternativity řídícího signálu IN jsou dány blokem podmíněného zápisu. Výraz tvaru /XXX/ → YYY představuje přenos obsahu registru XXX do registru YYY. Operandy se kterými budeme v algoritmu pracovat označíme H (integrační krok skládající se z mantisy MAH a exponentu $EXPH$) a operand Y (mantisa MAY a exponent $EXPY$).

Zápis mantisy operandu Y do registru RV a exponentu do RVE.

```
1. MAY→ SIR
2. /SIR/ → ACC, RV, SR
3. 0 → ACC
4. EXPY→ SIR
5. /SIR/ → ACC, RVE
```

Zápis hodnoty integračního kroku H. Mezitím proveden výpočet exponentu součinu $H * Y$ na řádku 7.

```
6. EXPH → SIR
7. /SIR/ + /ACC/ → ACC, PRI
8. MAH → SIR
9. 0 → ACC
```

Výpočet hodnoty mantisy součinu $H * Y$. Určení rozdílu mezi řády operandů a zápis znaménka výsledku rozdílu do klopného obvodu typu D v obvodu podmíněného zápisu PC.

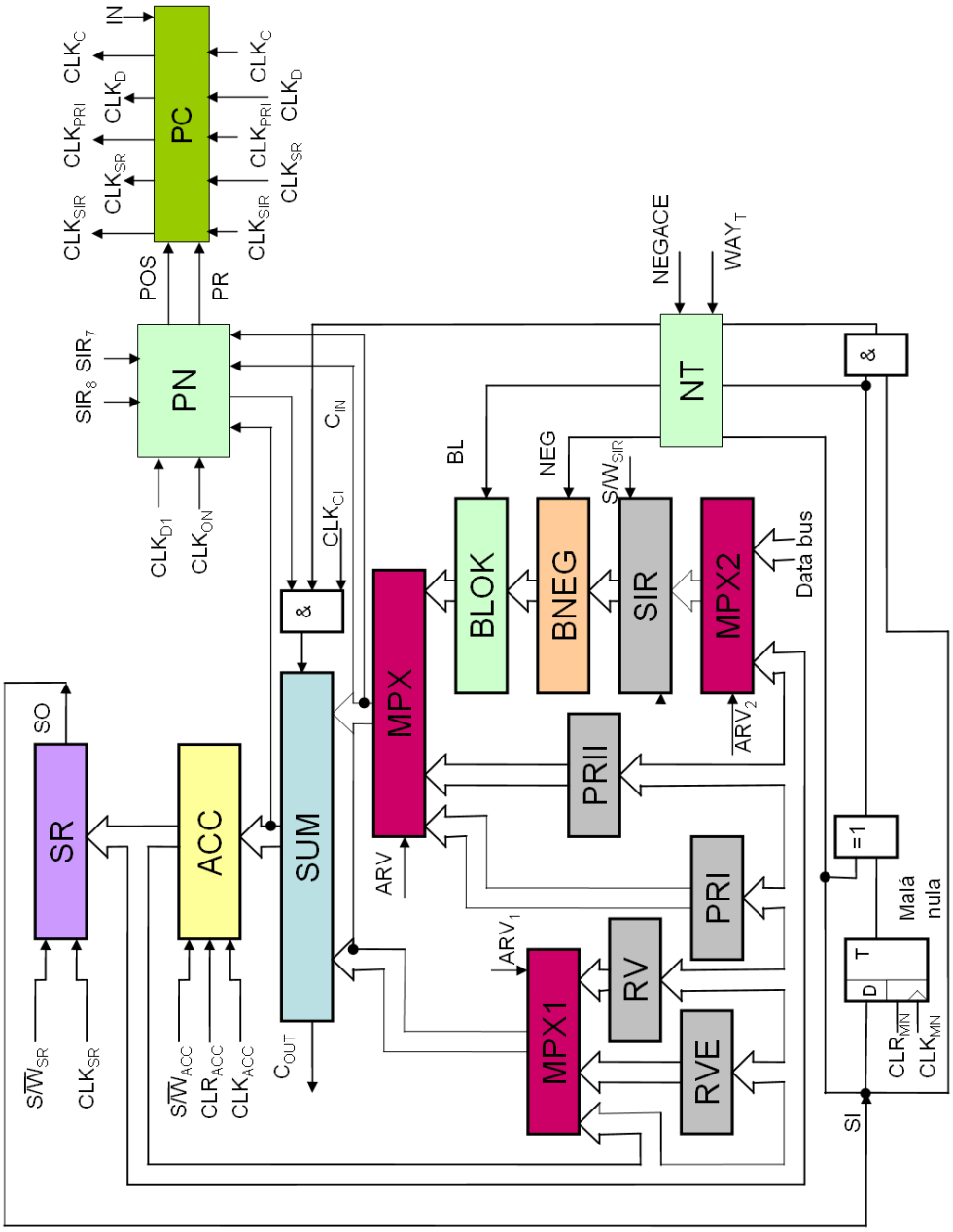
```
10. /SIR/ * /SR/ → ACC
11. /ACC/ → PRII
12. 0 → ACC
13. /PRI/ → ACC, SIR
14. NEG/SIR/ + 1 + /RVE/ → ACC
15. ACC_S → D klopný obvod
```

16. 0 → PRI
17. /ACC/ → SIR
18. /RVE/ → ACC

Úprava mantisy příslušného operandu pro sečtení s hodnotou Y (nastavení odpovídajících řádů). Řídící signál IN určuje, které operace se uskuteční.
19. $IN = 1$; /ACC/ + NEG/SIR/ + 1 → ACC, RVE
20. $IN = X$; 0 → ACC
21. $IN = 0$; NEG/SIR/ + 1 → ACC, PRI
22. $IN = X$; 0 → ACC
23. $IN = 0$; /PRII/ → ACC, SIR
24. $IN = X$; 0 → ACC
25. $IN = 0$; /PRI/ → ACC
26. $IN = X$; 0 → PRI
27. $IN = 0$; /ACC/ + 1 → ACC ↩
28. $IN = 0$; /RV/ → ACC
29. $IN = 1$; /SIR/ + /ACC/ → ACC, PRI
30. $IN = X$; 0 → ACC
31. $IN = 1$; /ACC/ → SIR
32. $IN = 1$; /RV/ → ACC, SIR
33. $IN = 1$; NEG/SIR/ + 1 + /ACC/ → ACC
34. $IN = 1$; /PRI/ → ACC
35. $IN = 1$; /ACC/ + 1 → ACC ↩
36. $IN = 1$; /PRII/ → ACC
37. $IN = 0$; CLR_D

Výpočet mantisy výsledku.
38. /SIR/ + /ACC/ → ACC
39. CLK_{D1}
40. 0 → PRI
41. ošetření přetečení a normalizace - viz. 3.4.1

Za pomoci registru SIR je nahrána hodnota mantisy Y do ACC, RV a SR. Poté dojde k anulování ACC a nahrání exponentu operandu Y do SIR. Odtud je hodnota uložena i do ACC a registru RVE. Nyní dojde k nahrání exponentu kroku H do SIR a sečtení s hodnotou v ACC. Výsledek je uložen do pomocného registru PRI. Následuje nahrání mantisy kroku H a proběhne násobení Boothovým algoritmem s uložením výsledku do pomocného registru PRII. Na řádku 14 je vypočítán rozdíl exponentů a poté je uloženo znaménko tohoto rozdílu. Pokud byl rozdíl exponentů záporný, získáme exponent konečného výsledku přičtením rozdílu exponentů k součtů exponentů z řádku 7 (řádek 19). Jinak je exponentem výsledku hodnota v RVE (EXPY). Poté dle znaménkového bitu je nahrána mantisa menšího z operandů k součtu a je proveden její logický posuv vpravo. Na řádku 27 pro mantisu výsledku násobení, popřípadě na řádku 35 pro mantisu vstupního operandu Y. Po korekci je provedeno sečtení a mantisa výsledné hodnoty je uložena v registrech SR, RV a exponent se nachází v registru RVE. Poté dojde k případnému ošetření přetečení a normalizaci.



Obrázek 3.15: Numerický integrátor pro práci s čísly v plovoucí řádové čárce.

3.4.3 Návrh a popis činnosti integrátoru zpracovávající mantisu a exponent paralelně

V kapitole 3.4.2 byl popsán numerický integrátor pracující v aritmetice pohyblivé řádové čárky, který zpracovává mantisu i exponent v jedné aritmetické jednotce. Druhou možností je současné zpracování.

Paralelní přístup vyžaduje dvě jednotky, kdy v první je prováděn výpočet mantis a druhá provádí výpočty nad exponenty. K tomu je zapotřebí dvou sběrnic - pro mantisu a exponent. Aritmetická jednotka exponentu musí zajistit součet i rozdíl exponentů, inkrementaci a řízení aritmetického posuvu mantisy. K tomu je zapotřebí napojení signálu posuvu S_{SIR} i k řadiči exponentu. Jednotka mantisy musí umožňovat násobení, sčítání, odečítání a aritmetické i logické posuvy. Schéma jednotlivých jednotek je uvedeno na obrázku 3.16 (mantisa) a 3.17 (exponent). Obě jednotky musí být synchronizovány tak, aby bylo dosaženo správného výpočtu. Jedná se především o synchronizaci před aritmetickým posuvem mantis, kdy výpočet v jednotce exponentu musí být synchronizován s výpočtem násobení mantis a poté synchronizaci samotného posuvu mantisy menšího operandu se vzájemnou inkrementací (budeme posouvat dle záporného rozdílu exponentů) rozdílu exponentů až po nulu.

Činnost numerického integrátoru mantisy

Algoritmus výpočtu nad mantisami je uveden v následujících příkazech. Operandy jsou označeny MAY a MAH, výraz tvaru /XXX/ → YYY představuje přenos obsahu registru XXX do registru YYY.

1. MAY→ SIR
2. /SIR/ → ACC, RV, SR
3. MAH → SIR
4. 0 → ACC
5. /SIR/ * /SR/ → ACC
6. /ACC/ → PRI
7. 0 → ACC
8. Synchronizace jednotek
9. $IN = 0$; /PRI/ → ACC, SIR
10. Synchronizace jednotek
11. Logický posuv vpravo hodnoty v registru SIR
12. 0 → ACC
13. $IN = 1$; /RV/ → ACC, SIR
14. Synchronizace jednotek
15. Logický posuv vpravo hodnoty v registru SIR
16. 0 → ACC
17. $IN = 0$; /RV/ → ACC
18. $IN = 1$; /PRI/ → ACC
19. /SIR/ + /ACC/ → ACC

Na řádcích 1 - 4 se načte mantisa operandu Y a H. Na řádku 5 se provede součin operandů a uložení hodnoty do registru PRI (řádek 6). Nastává první synchronizační bod, neboť další řízení jednotky je ovlivněno znaménkovým bitem rozdílu exponentů z jednotky

zpracovávající exponenty. Je-li znaménkový bit roven 0, posouvá se hodnota z registru PRI. Jinak je posouvána hodnota MAY. Posuvy musí být taktéž synchronizovány, neboť v jednotce exponentu paralelně s každým posuvem dochází k inkrementaci čítací hodnoty. Na řádku 19 získáme mantisu výsledku.

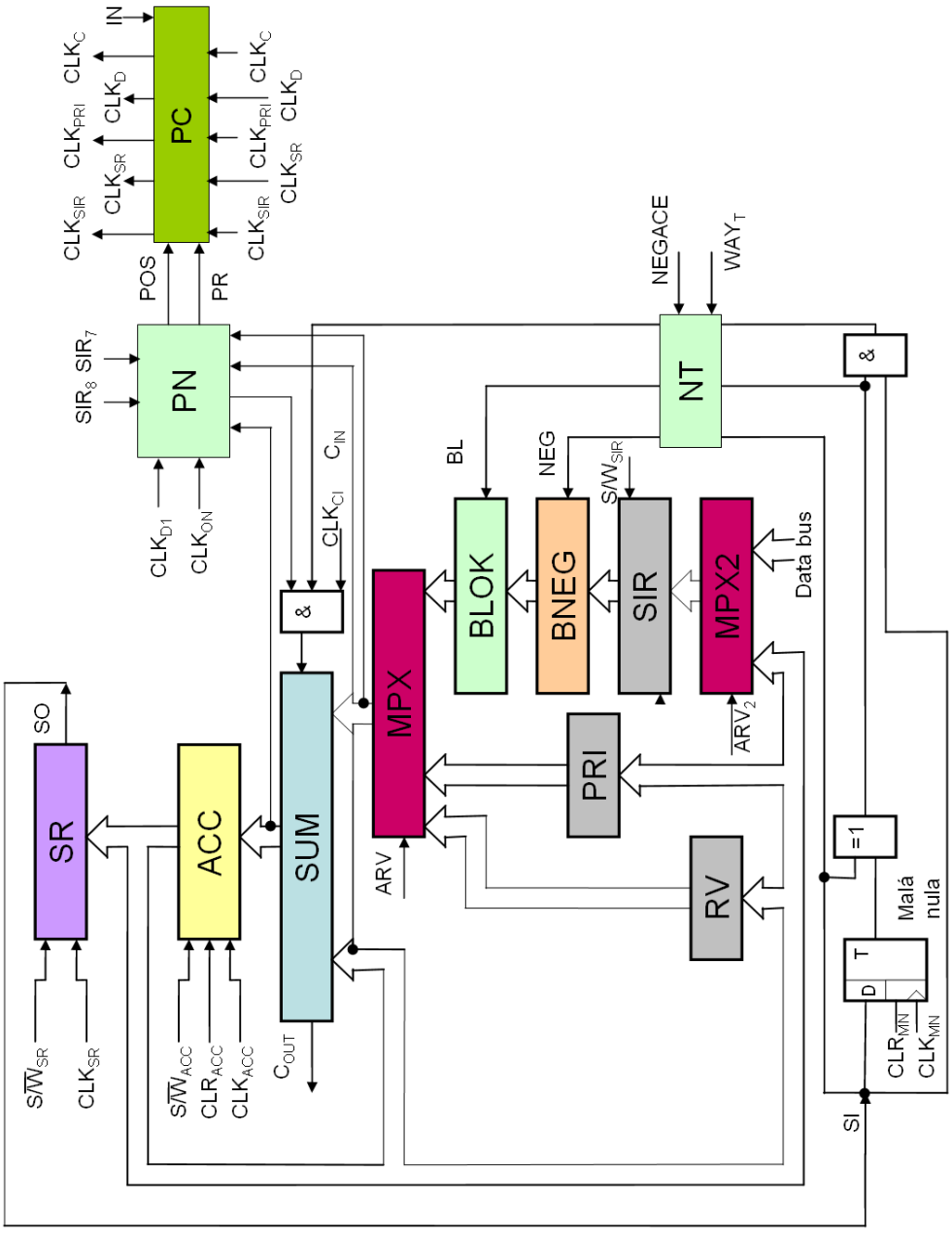
Činnost numerického integrátoru exponentu

Následuje zápis algoritmu, jež je prováděn v numerické jednotce exponentu. Zápis operací se drží již zavedených konvencí.

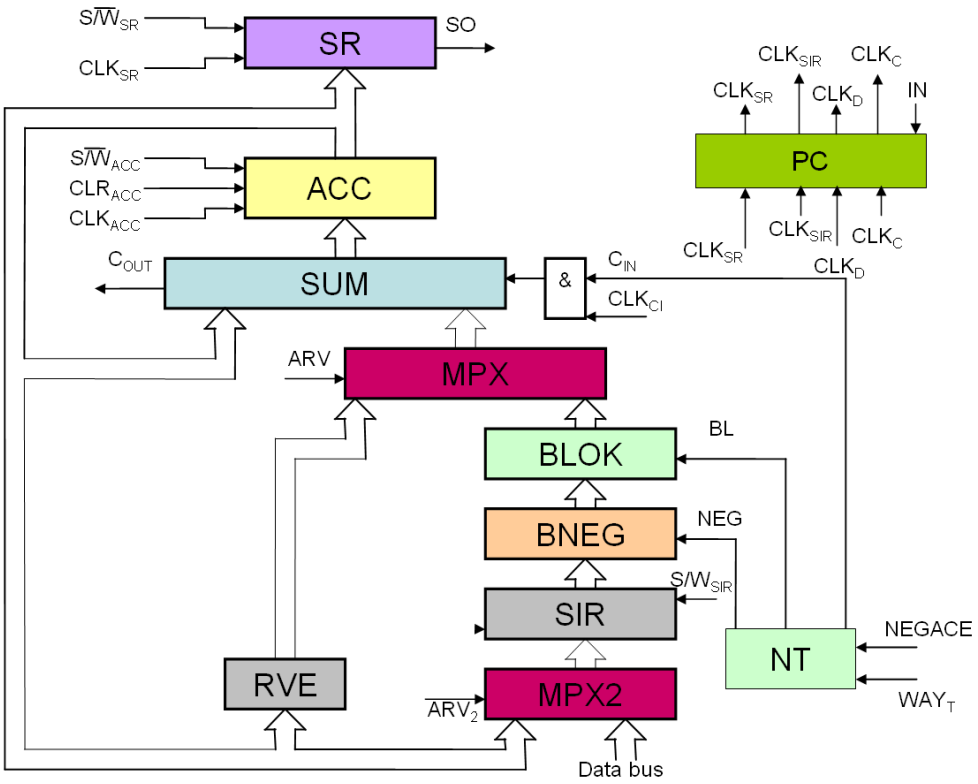
```
1. EXPY→ SIR
2. /SIR/ → ACC, RVE
3. EXPH → SIR
4. /SIR/ + /ACC/ → ACC, SIR
5. 0 → ACC
6. /RVE/ → ACC
7. NEG/SIR/ + 1 + /ACC/ → ACC, SIR
8. ACC_S → D klopný obvod
9. 0 → ACC
10. /RVE/ → ACC
11. IN = 1; /ACC/ + NEG/SIR/ + 1 → ACC, RVE
12. 0 → ACC
13. IN = 0; NEG/SIR/ + 1 → ACC
14. Synchronizace
15. IN = 0; /ACC/ + 1 → ACC ↩
16. 0 → ACC
17. IN = 1; /SIR/ → ACC
18. Synchronizace
19. IN = 1; /ACC/ + 1 → ACC ↩
```

V prvních třech krocích dojde k nahrání exponentů do registrů a ve čtvrtém kroku je vypočítán jejich součet. Na řádku 7 je vypočítán rozdíl exponentu $EXPY$ a hodnoty z řádku 4. Znaménko rozdílu je uloženo a na řádku 11 je rozdíl přičten k součtu exponentů, pokud byl rozdíl záporný. Tím je dán exponent výsledku, který je uložen po zbytek výpočtu v registru RVE. Pokud byl rozdíl kladný, je za pomoci doplňkového kódu převeden na zápornou hodnotu (řádek 13). Na řádcích 15 a 19 je prováděna inkrementace rozdílu, která je synchronizována s postupným logickým posuvem registru SIR jednotky mantisy.

Po skončení výpočtu se mantisa výsledku nachází v registru ACC jednotky mantisy a exponent v registri RVE jednotky zpracovávající exponent.



Obrázek 3.16: Numerický integrátor mantisy pracující s čísly v plovoucí řádové čárce.



Obrázek 3.17: Numerický integrátor exponentu pracující s čísly v plovoucí řádové čárce.

3.5 Numerický integrátor na principu metody Taylorovy řady

Dosavadní numerické integrátory jsme navrhovali s ohledem na použití Eulerovy metody. Tato metoda však neposkytuje potřebnou přesnost. Přesnější hodnoty lze získat za použití více členů Taylorovy řady:

$$y_{i+1} = y_i + \frac{h}{1!}y_i' + \frac{h^2}{2!}y_i'' + \frac{h^2}{2!}y_i''' + \cdots \tag{3.6}$$

Náš integrátor řeší následující rovnici:

$$y' = y \qquad y(0) = y_0 \tag{3.7}$$

Po dosazení rovnice 3.7 do 3.6 dostáváme:

$$Y_{i+1} = Y_i + DY1_i + DY2_i + DY3_i + \cdots \tag{3.8}$$

kde:

$$\begin{aligned} DY1_i &= hY_i \\ DY2_i &= \frac{h}{2}DY1_i \\ DY3_i &= \frac{h}{3}DY2_i \\ &\vdots \\ DYk_i &= \frac{h}{k}DY(k-1)_i \end{aligned}$$

Z rovnic vidíme, že rozšíření Eulerovy metody o výpočet dalších členů Taylorova rozvoje nepřináší žádné další operace. Numerický integrátor bude muset implementovat operace sčítání, odečítání a násobení. Tyto operace budeme provádět stejně jako u návrhů předchozích. Násobení bude prováděno Booth algoritmem a operace se rozpadnou na počítání s mantisami a exponenty. Během výpočtu si budeme muset poznamenat mantisu a exponent hodnoty DYk_i, abychom mohli na jejím základě vypočítat $DY(k+1)_i$. Výsledek budeme postupně zpřesňovat přičítáním dalších hodnot $DY(k+1)_i$. Proto i mantisu a exponent současného výsledku iterace budeme muset ukládat abychom k němu mohli přičíst hodnotu nově vypočítaného $DY(k+1)_i$. Ukládaných hodnot je více a pro jejich uložení nepoužijeme jednotlivé registry, jako doposud, ale použijeme paměť.

3.5.1 Návrh a popis činnosti integrátoru zpracovávající mantisu a exponent v jedné společné jednotce

Návrh a algortimus výpočtu bude velmi podobný návrhu v kapitole 3.4.2. Celkové zapojení numerického integrátoru postaveného na principu Taylorovy řady pracujícího s čísly v plovoucí řádové čárce je uvedeno na obrázku 3.18. Vidíme, že místo pomocných registrů RV, RVE, PRI, PRII je použit pro uložení hodnot paměťový prvek.

Činnost numerického integrátoru

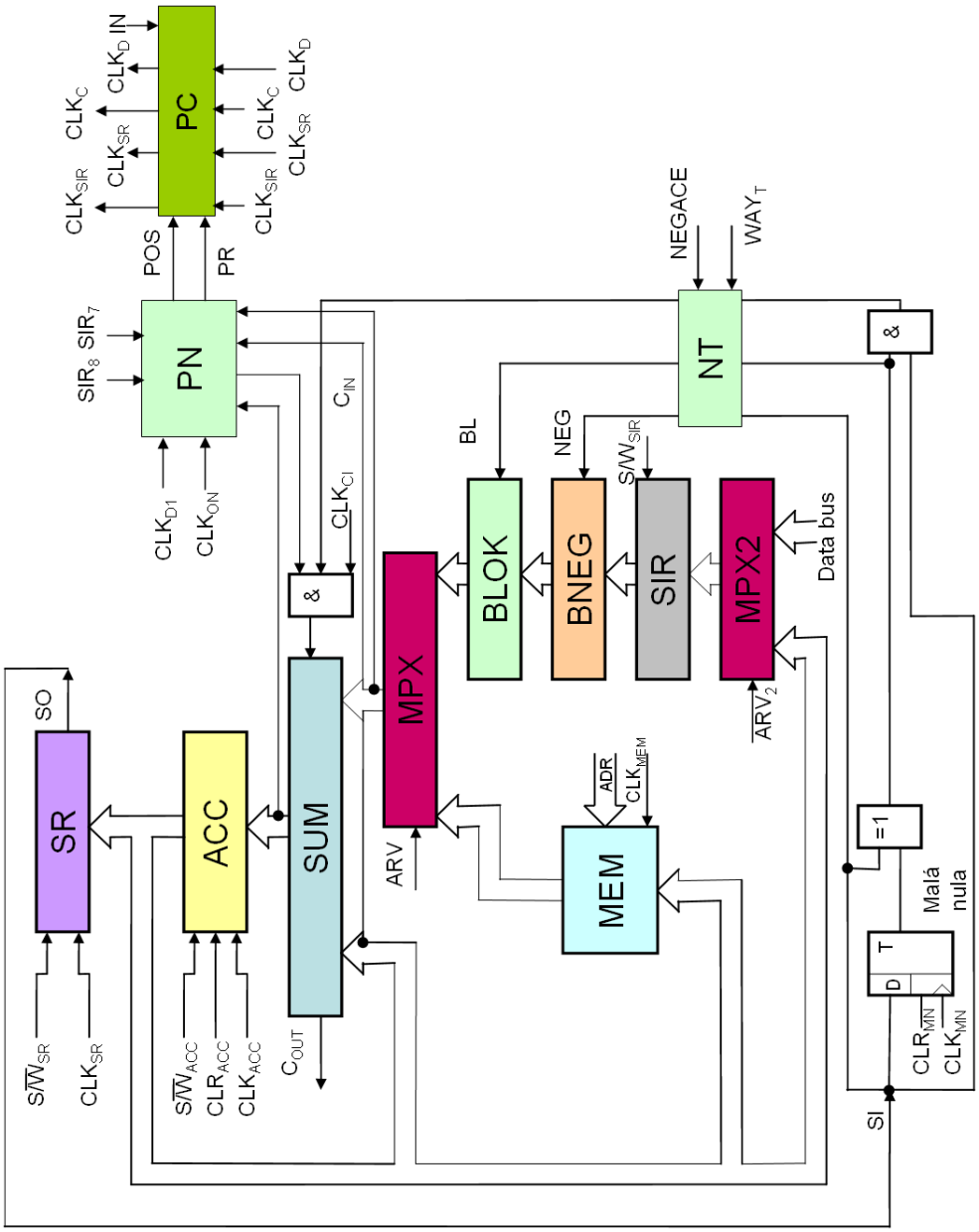
Algoritmus výpočtu si ukážeme na následující posloupnosti příkazů. Zápisy pro alternativity řídícího signálu IN jsou dány blokem podmíněného zápisu. Výraz tvaru /XXX/ → YYY představuje přenos obsahu registru XXX do registru YYY, popřípadě přenos hodnoty z/do paměti. Paměť budeme adresovat za pomoci prefixu M (MRVE, MRV, MPRI, MPRII,...). Operandy se kterými budeme v algoritmu pracovat označíme H (integrační krok skládající se z mantisy MAH a exponentu $EXPH$) a operand Y (mantisa MAY a exponent $EXPY$).

1. MAYO→ SIR
2. /SIR/ → ACC, MRV, SR
3. 0 → ACC
4. EXPYO→ SIR
5. /SIR/ → ACC, MRVE
6. EXPH → SIR
7. /SIR/ + /ACC/ → ACC, MPRI
8. MAH → SIR
9. 0 → ACC
10. /SIR/ * /SR/ → ACC
11. /ACC/ → MPRII
12. 0 → ACC
13. /MPRI/ → ACC, SIR
14. 0 → ACC
15. /MRVE/ → ACC
16. NEG/SIR/ + 1 + /ACC/ → ACC, MPO
17. ACC_S → D klopný obvod
18. /ACC/ → SIR
19. 0 → ACC
20. /MRVE/ → ACC
21. $IN = 1$; /ACC/ + NEG/SIR/ + 1 → ACC, MRVE
22. $IN = X$; 0 → ACC
23. $IN = 0$; NEG/SIR/ + 1 → ACC, MPO
24. $IN = X$; 0 → ACC
25. $IN = 0$; /MPRII/ → ACC, SIR
26. $IN = X$; 0 → ACC
27. $IN = 0$; /MPO/ → ACC
28. $IN = 0$; /ACC/ + 1 → ACC ↩
29. $IN = 0$; /MRV/ → ACC
30. $IN = 1$; /MPO/ → ACC
31. $IN = 1$; /ACC/ → SIR
32. $IN = 1$; NEG/SIR/ + 1 + /ACC/ → ACC
33. $IN = 1$; /MPO/ → ACC
34. $IN = 1$; /ACC/ + 1 → ACC ↩
35. $IN = 1$; /MPRII/ → ACC
36. $IN = 0$; CLR_D
38. /SIR/ + /ACC/ → ACC
39. CLK_{D1}
40. ošetření přetečení a normalizace - viz. 3.4.1

```
41. konec výpočtu? ano - GO TO 47, ne - GO TO 42
42. /ACC/ → MRV
43. 0 → ACC
44. /MPRII/ → SR
45. /MPRI/ → ACC
46. GO TO 6
47. END
```

Za pomoci registru SIR je nahrána hodnota mantisy počáteční podmínky Y_0 do ACC, SR a do paměti na adresu MRV. Poté dojde k anulování ACC a nahrání exponentu operandu Y_0 do SIR. Odtud je hodnota uložena i do ACC a MRVE. Nyní dojde k nahrání exponentu kroku H do SIR a sečtení s hodnotou v ACC. Výsledek je uložen do paměti na adresu MPRI. Následuje nahrání mantisy kroku H a proběhne násobení Boothovým algoritmem s uložením výsledku do MPRII. Díky uložení mantisy na adresu MPRII a exponentu na MPRI se může v další iteraci provést násobení krokem. Krok H je vždy nahrán předpřipraven. Tedy v první iteraci je nahrán krok H, v další $\frac{H}{2}$ a tak dále. Je tedy nutné mít tyto kroky předpočítány. Na řádku 16 je vypočítán rozdíl exponentů a poté je uloženo znaménko tohoto rozdílu. Řádek 21 obsahuje přičtení rozdílu exponentů k výsledku součtů exponentů z řádku 7, za předpokladu, že je rozdíl záporný ($IN = 1$), získáme exponent konečného výsledku. Jinak je výsledným exponentem hodnota v RVE. Poté dle znaménkového bitu je nahrána mantisa menšího z operandů k součtu a je proveden její logický posuv vpravo. Na řádku 28 pro mantisu výsledku násobení, popřípadě na řádku 34 pro mantisu operandu Y. Na řádcích 38 - 40 se sečte hodnota nově vypočítaného $DY(k+1)_i$ s předešlými součty. Pokud je výpočet u konce, je proveden skok na konec. V opačném případě je kumulovaný součet uložen do paměti na adresu MRV (řádek 42) a na řádcích 44 - 45 se uskutečňuje přesun hodnoty DYk_i do registrů SR (mantisa) a ACC (exponent) a pokračuje se výpočtem hodnoty $DY(k+1)_i$.

Vidíme, že postup výpočtu je velmi podobný s numerickým integrátorem pracujícím na principu Eulerovy metody. U Taylorovy řady je větší režie kvůli absenci registrů a proto lze provádět všechny aritmetické operace pouze s hodnotou načtenou z paměti v akumulátoru. Ukončení výpočtu lze například provést uložením průběžné hodnoty do pomocného registru (popřípadě místo v paměti) a srovnáním nové hodnoty. Pokud se nebudou lišit, lze výpočet ukončit. Popřípadě budou-li se lišit o danou hodnotu menší než přesnost. Jelikož se jedná o rychlé hardwarové operace, lze také provádět vždy přesný počet iterací, neboť režie kolem ukládání, srovnávání a porovnávání v každém kroku může být v některých případech delší.



Obrázek 3.18: Numerický integrátor pracující s čísly v plovoucí řádové čárce postavený na metodě Taylorovy řady.

3.5.2 Návrh a popis činnosti integrátoru zpracovávající mantisu a exponent paralelně

V kapitole 3.4.3 jsme uvedli návrh jednotky zpracovávající mantisu a exponent ve dvou jednotkách paralelně. Tento návrh na principu Eulerovy metody v této kapitole, na základě poznatků uvedených v 3.5, rozšíříme o možnost výpočtu více členů Taylorovy řady.

Činnost numerického integrátoru mantisy

Algoritmus výpočtu nad mantisami je uveden v následujících příkazech. Operandy jsou označeny MAY a MAH, výraz tvaru /XXX/ → YYY představuje přenos obsahu registru XXX do registru YYY, popřípadě přenos hodnoty z/do paměti. Paměť budeme adresovat za pomoci prefixu M (MRV, MPRII,...). Celkové zapojení je na obrázku 3.19.

1. MAY0→ SIR
2. /SIR/ → ACC, MRV, SR
3. MAH → SIR
4. 0 → ACC
5. /SIR/ * /SR/ → ACC
6. /ACC/ → MPRII
7. 0 → ACC
8. Synchronizace jednotek
9. $IN = 0$; /MPRII/ → ACC, SIR
10. Synchronizace jednotek
11. Logický posuv vpravo hodnoty v registru SIR
12. 0 → ACC
13. $IN = 1$; /MRV/ → ACC, SIR
14. Synchronizace jednotek
15. Logický posuv vpravo hodnoty v registru SIR
16. 0 → ACC
17. $IN = 0$; /MRV/ → ACC
18. $IN = 1$; /MPRII/ → ACC
19. /SIR/ + /ACC/ → ACC
20. Synchronizace
21. ošetření přetečení a normalizace - viz. 3.4.1
22. konec výpočtu? ano - GO TO 27, ne - GO TO 23
23. /ACC/ → MRV
24. 0 → ACC
25. /MPRII/ → SR
26. GO TO 3
27. END

Na řádcích 1 - 4 se načte mantisa operandu $Y0$ a H. Na řádku 5 se provede součin operandů a uložení hodnoty do paměti na adresu MPRII (řádek 6). Nastává první synchronizační bod, neboť další řízení jednotky je ovlivněno znaménkovým bitem rozdílu exponentů z jednotky zpracovávající exponenty. Je-li znaménkový bit roven 0, posouvá se hodnota z MPRII. V opačném případě MAY. Posuvy musí být taktéž synchronizovány, neboť v jednotce exponentu paralelně s každým posuvem dochází k inkrementaci čítací

hodnoty. Na řádku 19 získáme mantisu výsledku. Pokud není výpočet u konce, provede se uložení aktuální sumy a nahraje se mantisa $DY(k)_i$ do registru SR. Další iterace probíhá od řádku 3, kdy se načte mantisa příslušného kroku operandu. Krok je tedy nutno mít znovu předpočítán.

Činnost numerického integrátoru exponentu

Následuje zápis algoritmu, jež je prováděn v numerické jednotce exponentu. Zápis operací se drží již zavedených konvencí. Ovodová realizace jednotky exponentu je na obrázku 3.20.

1. EXPY0→ SIR
2. /SIR/ → ACC, MRVE
3. EXPH → SIR
4. /SIR/ + /ACC/ → ACC, SIR, MPRI
5. 0 → ACC
6. /MRVE/ → ACC
7. NEG/SIR/ + 1 + /ACC/ → ACC, SIR, MPO
8. ACC_S → D klopný obvod
9. 0 → ACC
10. /MRVE/ → ACC
11. $IN = 1$; /ACC/ + NEG/SIR/ + 1 → ACC, MRVE
12. 0 → ACC
13. $IN = 0$; NEG/SIR/ + 1 → ACC, SIR
14. Synchronizace
15. $IN = 0$; /ACC/ + 1 → ACC ↩
16. 0 → ACC
17. $IN = 1$; /MPO/ → ACC
18. Synchronizace
19. $IN = 1$; /ACC/ + 1 → ACC ↩
20. Synchronizace
21. ošetření přetečení a normalizace - viz. 3.4.1
22. konec výpočtu? ano - GO TO 27, ne - GO TO 23
23. /ACC/ → MRVE
24. 0 → ACC
25. /MPRI/ → ACC
26. GO TO 3
27. 0 → ACC
28. /MRVE/ → ACC
29. END

V prvních třech krocích dojde k nahrání exponentů do registrů a ve čtvrtém kroku je vypočítán jejich součet. Na řádku 7 je vypočítán rozdíl exponentu $EXPY$ a hodnoty z řádku 4. Znaménko rozdílu je uloženo a na řádku 11 je rozdíl přičten k součtu exponentů, pokud byl rozdíl záporný. Tím je dán exponent výsledku, který je uložen po zbytek výpočtu v registru RVE. Pokud byl rozdíl kladný, je za pomoci doplňkového kódu převeden na zápornou hodnotu (řádek 13). Na řádcích 15 a 19 je prováděna inkrementace rozdílu, která je synchronizována s postupným logickým posuvem registru SIR jednotky mantisy. Není-li

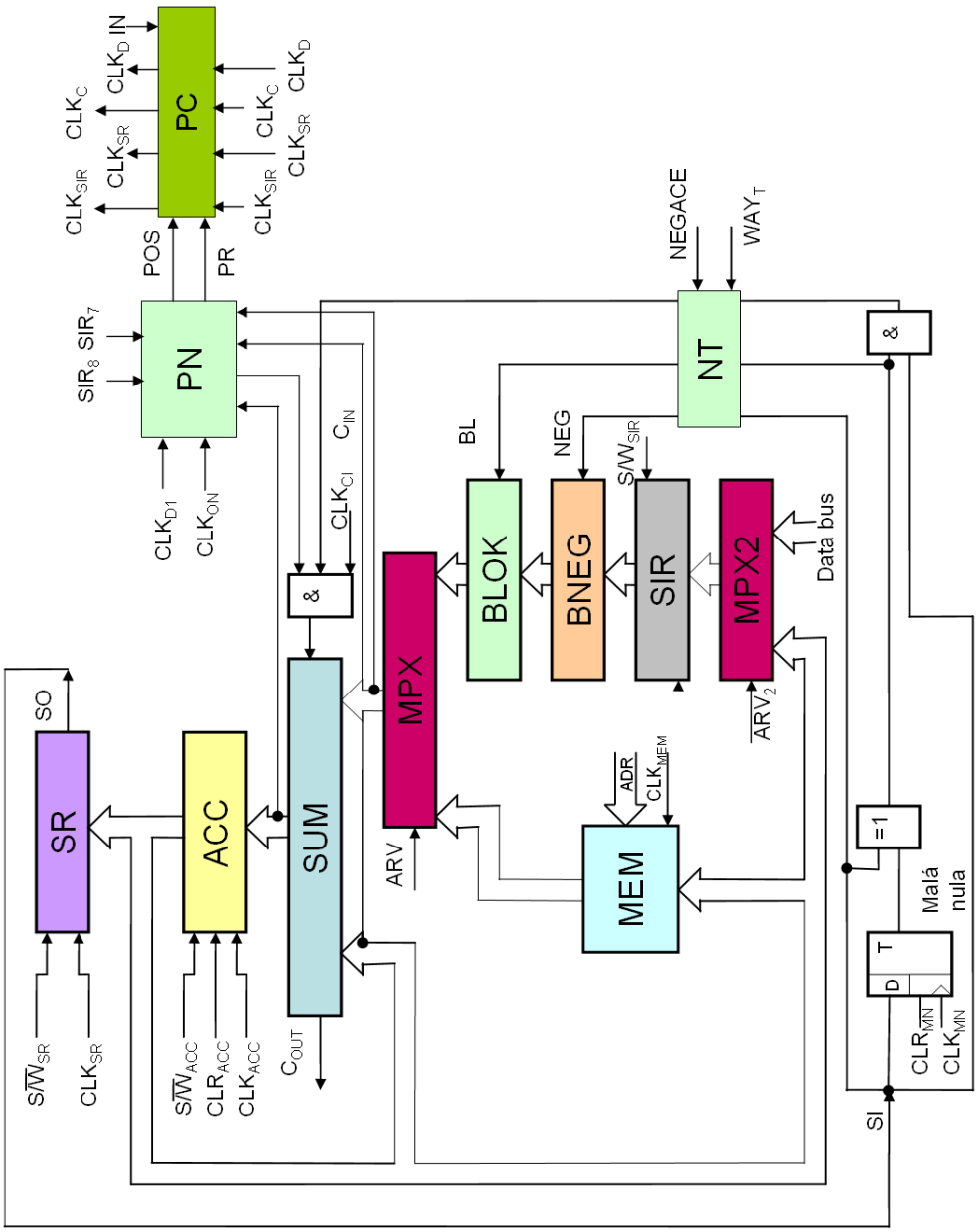
výpočet u konce, nahraje se exponent kumulovaného součtu do ACC a pokračuje se třetím řádkem.

3.6 Zhodnocení

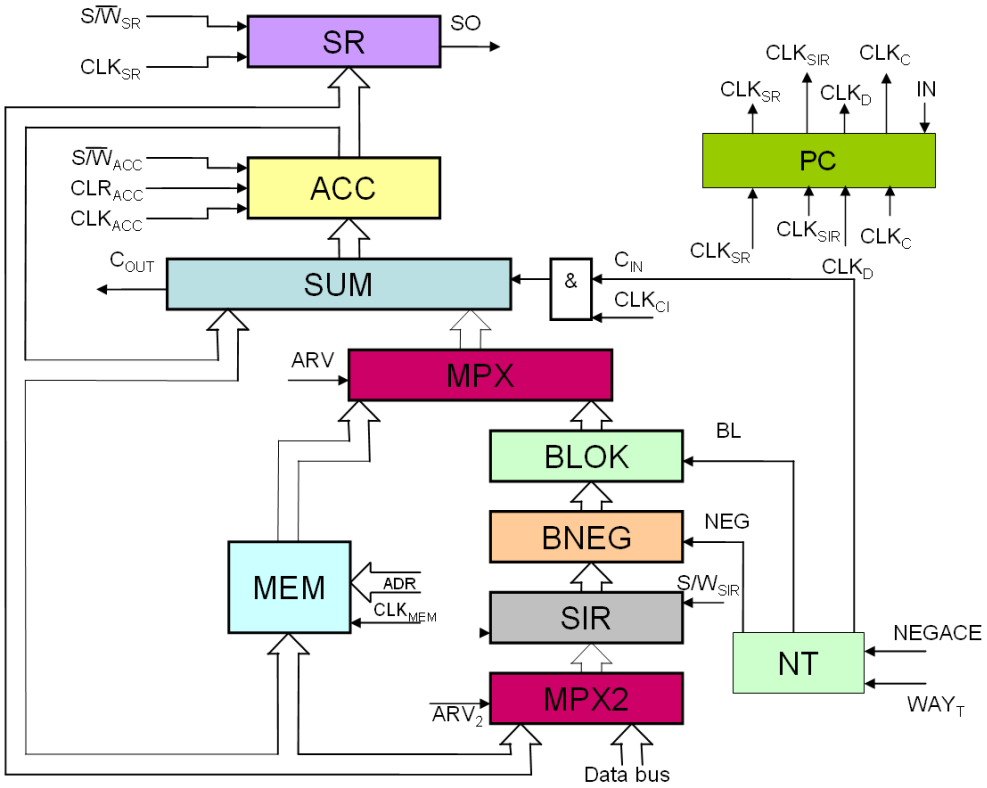
Kapitola nás nejprve seznámila s aritmetickými operacemi součtu, rozdílu a násobení s čísly v pevné a pohyblivé čárce, jež byly východiskem k návrhu numerických integrátorů. Jednotku v pevné řádové čárce jsme rozšířili o prvky, které jsou nutné ke zvládnutí operací nad čísly v plovoucí řádové čárce, jakou jsou obvod ošetření přetečení, obvod podmíněného zápisu, obvod normalizace a další. Za pomoci těchto prvků jsme sestavili numerické integrátory pracující na principu Eulerovy metody. Nejprve návrh pracující s mantisou a exponentem postupně. Tento algoritmus výpočtu jsme analyzovali a identifikovali jsme paralelní části výpočtu. Díky tomu jsme navrhli numerický integrátor skládající se ze dvou dílčích jednotek. Jednotky mantisy a exponentu, které pracují paralelně. Paralelní přístup si však vyžádal dodatečnou režii v podobě synchronizace výpočtů.

Eulerova metoda však neposkytuje uspokojivou přesnost výpočtu. Provedli jsme tedy nástin nutných modifikací obou přístupů (paralelního a sériového zpracování mantis a exponentů) k implementaci výpočtu za pomocí více členů Taylorova rozvoje.

Díky uvedeným návrhům na řešení numerického integrátoru získáváme přehled o výhodnosti a náročnosti na implementaci jednotlivých variant. V dalším textu se zaměříme na variantu plovoucí řádové čárky postavené na Eulerově metodě, neboť právě tento návrh je východiskem pro případné rozšíření o výpočet dalších členů Taylorovy řady a tím i zpřesnění numerické integrace.



Obrázek 3.19: Numerický integrátor mantisy pracující s čísly v plovoucí řádové čárce postavený na metodě Taylorovy řady.



Obrázek 3.20: Numerický integrátor exponentu pracující s čísly v plovoucí řádové čárce postavený na metodě Taylorovy řady.

Kapitola 4

Simulace

V rámci této práce provedeme simulaci návrhu numerického integrátoru v plovoucí řádové čárce pracující na principu Eulerovy metody. A to varianty integrátoru zpracovávající mantisu a exponent ve společné jednotce a návrh, jež zpracovává mantisu a exponent odděleně. Díky tomu budeme moci porovnat oba návrhy z pohledu počtu jednotlivých prvků, počtu operací, délky výpočtu a množství podpůrných instrukcí, které nepřímo souvisí s výpočtem. Například, jak bylo zmíněno v kapitole 3.4.3, numerický integrátor zpracovávající mantisu a exponent odděleně potřebuje synchronizaci ke vzájemné činnosti a proto se v algoritmu výpočtu využívají i prázdné operace. Naproti tomu sériové zpracování mantisy a exponentu ve společné jednotce bude mít svou režii v postupném načítání jednotlivých hodnot do registrů a postupném zpracování.

V této kapitole tedy ukážeme realizaci návrhů jednotek a provedeme stanovení statistických metod pro vzájemné porovnání. Na základě těchto metod provedeme srovnání a zhodnocení dosažených výsledků.

4.1 Návrh simulátoru

Cílem této práce je vytvoření simulátoru pro studijní účely, který realizujeme s co nejmenší závislostí na použitém operačním systému. Budeme-li chtít pracovat ve spojení s jazykem C++ máme několik možností. Můžeme například použít frameworky wxWidgets, U++ popřípadě Qt. Poslední zmiňovaný je komplexní framework, který poskytuje pro vývoj vlastní IDE, včetně grafického pro přímý návrh GUI, které lze snadno exportovat do xml struktůry. Z tohoto důvodu byl zvolen pro implementaci simulátoru. Qt ve verzi 4 je přenosný mezi unixovými operačními systémy (Linux, BSD), Windows a Mac OS X. Tím by měla být zajištěna přenositelnost hotového simulátoru při dodržení použití standartních postupů. Samotný framework je napsán v jazyce C++ ale podporuje i další jazyky. My se však budeme držet Qt ve spojení s jazykem C++.

Nyní se zaměřme na to, co by v tomto simulátoru mělo být simulováno. Jak jsme nastínili v úvodu kapitoly, budeme srovnávat realizace numerického integrátoru v plovoucí řádové čárce a to přesněji realizace zpracovávající mantisu a exponent v jedné a ve dvou oddělených jednotkách. Bude tedy nutné využívat stejné operace v obou integrátorech abychom mohli provést srovnání. Potřebné operace jsou dány algoritmy výpočtu pro jednotlivé realizace (numerický integrátor zpracovávající mantisu a exponent ve společné jednotce zde 3.4.2, numerický integrátor zpracovávající mantisu a exponent paralelně ve dvou oddělených jednotkách zde 3.4.3). Budeme tedy potřebovat přesouvat hodnoty mezi registry (**move**),

načítat hodnoty z adresové sběrnice do registru (**load**), pro realizaci Boothova algoritmu je nutno provádět aritmetický (**sha**) a logický posuv vpravo (**shr**) v registrech ACC, SR a SIR, nulovat hodnoty (**rst**). Dále bude nutno nastavovat řídící signály u všech multiplexorů, dílčí hodinové signály a speciální signály jako signál IN či povolení činnosti obvodu COND. Tyto signály budeme v simulátoru nastavovat prostřednictvím funkce **setSignal** a naopak jejich hodnoty budeme zjišťovat komplementární funkcí **getSignal**. V návrhu vidíme i logické členy. Jejich funkci nám budou realizovat funkce **AND**, **OR**, **EXOR** a **XNOR**. Sčítání bude realizováno funkcí **sum**, která sečte dvě vstupní hodnoty sumátoru, popřípadě přičte přenos v podobě C_{in} a výsledek uloží do SUM. Abychom mohli simulovat vzájemné čekání na zpracování mantisy a exponentu v oddělených jednotkách, je nutno mít v simulátoru instrukci prázdné operace (**nop**).

Námi uvedené operace zapouzdřují v reálném prostředí množinu elementárních operací. Budeme-li například přesouvat hodnotu mezi dvěma registry, musí dojít k nastavení zápisu hodnoty na sběrnici, nastavení signálu $WRITE$ vstupního registru a s náběžnou hranou hodinového signálu dojde k samotnému zápisu hodnoty do jednotlivých D-klopných obvodů registru. Pro naše potřeby je však taková míra přesnosti a detailnosti zbytečná. Budeme-li porovnávat vždy společné operace mezi oběma realizacemi, můžeme tyto elementární operace ponechat abstrahované. Pro studijní účely principu simulátoru je také zbytečné zacházet do hardwarových detailů realizace jednotlivých operací. Dodržíme však elementární zásady dáné návrhem a přesun hodnot budeme realizovat vždy mezi sousedními registry, stejně jako negaci budeme provádět za pomocí registru BNEG a přičtení 1 v podobě C_{in} na vstupu sumátoru. Podobně i ostatní operace se budou držet návrhů uvedených v předchozích kapitolách.

4.2 Návrh GUI

Simulátor bude obsahovat tři hlavní obrazovky mezi kterými se uživatel přepne za pomocí hlavního menu. První pohled (obrázek B.1) bude obsahovat numerický integrátor, jehož schéma je uvedeno na obrázku 3.15. Jedná se tedy o integrátor zpracovávající mantisu a exponent ve společné jednotce. Vlevo od obvodové realizace bude informační okno obsahující prováděné instrukce. Pod ním budou umístěna obslužná tlačítka. Další obrazovka simulátoru bude mít stejné rozložení, jen obvodová realizace jedné společné jednotky pro mantisu a exponent bude nahrazena jednotkami zpracovávající mantisu a exponent zvlášť (obrázek B.2). Obě jednotky záměrně umístíme na jeden pohled, neboť pracují zároveň a jsou propojeny (viz 3.4.3). Tímto rozložením je i dána velikost okna celého simulátoru, které by nemělo přesáhnout XGA rozlišení kvůli případným prezentacím za pomoci projektoru. Poslední okno programu bude obsahovat statistiky jednotlivých operací uvedených v 4.1. Jednotlivá okna budou přístupná během výpočtů a bude možno mezi nimi přepínat. Tím lze simultálně porovnávat kroky výpočtu mezi jednotlivými obvodovými realizacemi se současným přístupem k aktuálním statistickým údajům.

4.3 Návrh metod pro porovnání realizací

Během simulace se počítá vykonání jednotlivých operací v jednotlivých návrzích. Jakým způsobem však výsledky interpretovat?

Nejprve můžeme srovnat oba přístupy co do počtu taktů, za kterých získáme výsledek. Tím získáme představu, zda-li zpracovávání mantisy a exponentu v oddělených jednotkách

je rychlejší a případně kolikrát. Na druhou stranu musíme u odděleného zpracování analyzovat, kolik prázdných synchronizačních operací bylo provedeno a jaký je poměr těchto operací k celkovému počtu. Analyzujeme i sumy jednotlivých operací a zamyslíme se nad jejich vztahem a poměrem k ostatním operacím. Vedle hodnocení instrukcí bude zajímavé zhodnotit registry z pohledu jejich využití. To stanovíme na základě změn hodnot do nich uložených.

Dále bychom se měli zabývat otázkou závislosti počtu kroků výpočtů na bitových délkách jednotlivých zpracovávaných hodnot a jak ovlivní zvětšení operandů počet jednotlivých instrukcí. Závěrem porovnáme obě řešení z hlediska počtu použitých prvků.

4.4 Implementace

Jednotlivé kroky výpočtu jsou realizovány příslušnými řídícími funkcemi. Numerický integrátor sériového řešení mantisy a exponentu má svůj řadič realizovaný funkcí **radicSpolecna**. Podobně mantisa numerického integrátoru paralelního řešení je řízená funkcí **radicOddelenaMantisa** a exponent **radicOddelenaExponent**. Tyto funkce volají postupně funkce uvedené v 4.1 dle zápisů algoritmů uvedených v kapitole 3.4. Operace násobení, jak jsme zmínili v 3.1.3, je realizována Boothovým algoritmem, který je prováděn funkcí **mul**. Pro sečtení operandů je nutné mít mantisy nastaveny na stejný řád exponentu. Mantisu menšího čísla je tak nutno posunout o rozdíl exponentů. Posunutí mantisy je realizováno funkcí **mantisaShift**. Posunutí mantisy v rámci paralelního zpracování mantisy a exponentu je třeba synchronizovat, kdy jednotka exponentu čeká na dokončení násobení Boothovým algoritmem v jednotce mantisy. Synchronizace je zajištěna synchronizačním blokováním kroku jednotky exponentu za pomocí operace **nop**. Ukončení čekání je provedeno jednotkou mantisy zvýšením kroku jednotky exponentu a díky tomu jednotka exponentu pokračuje ve svém zpracování. Podobně je zajištěna synchronizace samotného posouvání funkcí **shiftMantisa** kdy naopak jednotka mantisy čeká na dokončení posuvu řízeného z jednotky exponentu.

4.5 Výsledky simulace

Simulaci jsme prováděli postupně s operandy s bitovou délkou mantisy osm bitů a exponentem délky čtyři bity. Poté jsme obě hodnoty zdvojnásobili a odsimulovali jsme výpočet na pěti příkladech. Zadání příkladů najdeme v tabulce C.1. Dostali jsme tak deset statistických přehledů, které najdete v příloze C. Nejprve si rozeberme výpočty s kratšími operandy.

Počet operací **mem** nás jistě nepřekvapí neboť načítáme z adresové sběrnice právě čtyři hodnoty. Počet instrukcí **load**, pomoci kterých se načítá hodnota do registru SIR, je totožný s počtem změn hodnoty tohoto registru. Aritmetické posuvy jsou prováděny při násobení Boothovým algoritmem. Díky tomu, že všechny příklady 1. - 5. měly shodnou bitovou velikost, je počet posuvů shodný. Logické posuvy vpravo jsou mimo násobení(registr SR) prováděny nad mantisou menšího operandu před sečtením (v registru SIR). Počet těchto posuvů je při stejném rozdílu exponentů zcela totožný. Stejnou závislost lze vypozorovat i z počtu operací **sum**. Operace **nop** se v realizaci numerického integrátoru zpracovávající mantisu a exponent ve společné jednotce nevyskytuje.

Podíváme-li se na počet změn hodnot v registrech, vidíme, že registry PRII, PRI, RV, RVE, SR a SIR mají v každém výpočtu stejný počet změn. Do registr SR je na začátku výpočtu nahrána hodnota mantisy počáteční hodnoty, která je v průběhu násobení po-

souvána a kombinace sousedních bitů řídí Booth algoritmus (viz 3.1.3). Stejná hodnota je uložena během celého výpočtu v registru RV. Exponent výsledku násobení je nahrán do registru PRI a mantisa do registru PRII. Zde je uchována do případného posuvu o rozdíl exponentů a sečtení s mantisou počáteční podmínky. RVE obsahuje postupně hodnotu exponentu počáteční podmínky a je-li rozdíl exponentů při výpočtu posunutí záporný, je tento rozdíl přičten k RVE, čímž dostaneme exponent výsledku.

Srovnáme-li počet operací v numerickém integrátoru zpracovávající paralelně mantisu a exponent, zjistíme, že počet vykonaných instrukcí narostl oproti postupnému zpracování. V jednotce zpracovávající mantisu je prováděn Booth algoritmus. Platí zde stejné závislosti jako u numerického integrátoru zpracovávající exponent a mantisu ve společné jednotce. Počet změn hodnot registrů v jednotce mantisy je u všech příkladů konstantní. Do registru SR a RV je na začátku zpracování nahrána mantisa počáteční podmínky. Výsledek násobení je uložen do PRI. Jednotka simulující výpočet exponentu využívá jedinou operaci posuvu a tou je logický posuv vpravo. Signál posuvu je však napojen na jednotku mantisy a zde je posouvána mantisa menšího z operandů před operací sečtení v registru SIR. Počet posuvů je tedy přímo úměrný rozdílu exponentů. Jak vidíme z tabulky 4.1 jednotka exponentu čeká mnohem déle na výpočet mantisy (v průměru 31,73 % času). Díky konstatnímu času čekání, lze usuzovat, že je způsobeno dobýhajícím výpočtem Booth algoritmu v jednotce mantisy. Jednotka mantisy naopak čeká, než je provedena korekce mantisy menšího z operandů. A to průměrně 10,87 % výpočetního času. Z výsledků plyne, že zvětšení rozdílů exponentů o jedničku přidá 4 operace čekání.

Počet nop operací v jednotce (pět příkladů)					
mantisy			exponentu		
celkem operací	nop operací	% nop	celkem operací	nop operací	% nop
87	6	6.9 %	74	25	33.78 %
93	10	10.75 %	79	25	31.65 %
92	10	10.87 %	79	25	31.65 %
95	14	14.74 %	83	25	30.12 %
93	10	10.75 %	79	25	31.65 %

Tabulka 4.1: Doba vzájemného čekání jednotek při zpracování kratších operandů.

Výsledek ve společné jednotce je v průměru za 149 operací a u zpracování mantisy a exponentu ve zvláštních jednotkách dostaneme výsledek za 92 taktů. To je zrychlení paralelního algoritmu o 38,26 %. Zrychlení pro jednotlivé příklady najdeme v tab 4.2.

	sériová jednotka (taktů)	paralelní jednotka (taktů)	zrychlení (v %)
1. příklad	144	87	39.58 %
2. příklad	150	93	38.00 %
3. příklad	149	92	38.26 %
4. příklad	152	95	37.50 %
5. příklad	150	93	38.00 %

Tabulka 4.2: Zrychlení paralelního zpracování (menší operandy).

Nyní analyzujme příklady ve kterých vystupovaly operandy s dvojnásobnou bitovou

délkou mantis i exponentů. Jednotlivé údaje jsou taktéž uvedeny v příloze C. Celkově se zvedl počet operací díky Booth algoritmu a posunování mantis menšího z operandů. Na ostatní operace algoritmu nemělo zvětšení bitové délky vliv. Čekání jednotky exponentu na dokončení násobení mělo dopad na mnohonásobné zvětšení počtu prázdných operací. Stejně i případné posuvy mantisy, která se posouvala v řádech desítek posuvů, způsobila nárust čekání jednotky mantisy. Tabulka 4.3 udává průměrnou dobu 40,54 % času čekání jednotky mantisy a exponent je ve stavu nečinnosti po 35,22 % výpočetního času. Díky tomu je zrychlení paralelního zpracování mantisy s exponentem nižší (tabulka 4.4) než u kratších operandů. S čím delšími operandy tedy pracujeme, tím více převládá váha čekacích operací a výhody paralelního přístupu k mantise a exponentu se začínají ztrácet.

Počet nop operací v jednotce (pět příkladů)					
mantisy			exponentu		
celkem operací	nop operací	% nop	celkem operací	nop operací	% nop
257	122	47.47 %	239	73	30.54 %
153	18	11.76 %	134	73	54.48 %
217	82	37.79 %	198	73	36.87 %
249	114	45.78 %	231	73	31.60 %
344	206	59.88 %	323	73	22.60 %

Tabulka 4.3: Doba vzájemného čekání jednotek při zpracování delších operandů.

	sériová jednotka (taktů)	paralelní jednotka (taktů)	zrychlení (v %)
1. příklad	314	257	18.15 %
2. příklad	210	153	27.14 %
3. příklad	274	217	20.80 %
4. příklad	306	249	18.63 %
5. příklad	401	344	14.21 %

Tabulka 4.4: Zrychlení paralelního zpracování (větší operandy).

4.6 Zhodnocení

Cílem simulace bylo zhodnocení výhodnosti a efektivity jednotlivých realizací. Z výsledků plyne předpoklad, že paralelním přístupem dostaneme řešení v kratším časovém úseku. Paralelní řešení se však potýká s problémem synchronizace, která je nutná ve dvou místech výpočtu a tím je rychlost řešení závislá na bitových délkách operandů. Tato závislost je přímá. V praxi bude požadavek na přesné a rychlé řešení klíčový. Budou se tedy zpracovávat operandy minimálně s délkami danými v normě IEEE 754 [7]. Bude se tedy v průměru dosahovat výsledků, které jsou dány naší simulací s delšími operandy (neboť jsme uvožavali stejnou bitovou délku exponentu). Paralelní přístup tak bude poskytovat urychlení, které je ale vykoupeno počtem použitých prvků, potřebou dvou řadičů a nutností většího počtu propojovacích sběrnic.

Paralelní přístup k řešení výpočtu obsahuje navíc oproti sériovému jeden multiplexor, sumátor, blokovací a negovací obvod a čtyři registry. Obvod tak má požadavky na dvojná-

sobnou plochu a na dvojnásobné řešení jednotlivých datových sběrnic. Navíc musí být řízen dvěmi řídícími jednotkami. Je tak na zvážení, zda-li je výhodnější v praxi implementovat paralelní přístup. Přihlédneme-li k tomu, že těchto obvodů má být při řešení problémů uvedených v 2.1.4 zapojeno hned několik desítek a že s rostoucími požadavky na délku operandů se nám výhodnost paralelního řešení snižuje, je výhodnější přístup postupného zpracování mantisy a exponentu v jedné společné jednotce. Zapojení více jednotek díky menšímu požadavku na prostor na čipu je méně problematické, nemusí se zde provádět synchronizace a celková cena bude také výrazně nižší.

Kapitola 5

Paralelní systémy

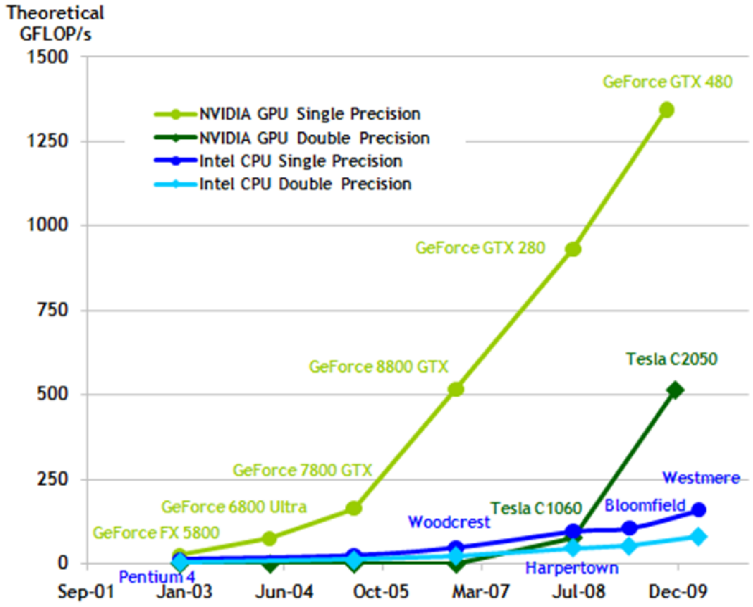
V úvodní kapitole jsme nastínili problém kmitající struny. Z analýzy problému jsme získali řešení v podobě soustavy diferenciálních rovnic 2.27 - 2.29. Tyto rovnice jsme řešili obvodovým schématem uvedeným na obrázku 2.3 a na základě tohoto schématu jsme zjistili potřebu integrátoru, který jsme v dalších kapitolách blíže rozvedli a navrhli numerické integrátory pracující s čísly v plovoucí řádové čárce. Vidíme, že se jedná o problém, jehož řešení je dáno rozsáhlými rovnicemi, kdy nad různými hodnotami provádíme shodné operace. Dostáváme tak přístup zvaný SIMD (single instruction multiple data).

5.1 Současné paralelní systémy

Pro rozsáhlé výpočetní problémy typu SIMD je paralelní přístup nejvýhodnější. V současné době máme mnoho nástrojů a přístupů, jak efektivně a relativně jednoduše realizovat paralelní výpočet. Během devadesátých let se zvyšoval počet FLOPs běžných CPU díky exponenciálnímu růstu zapojených tranzistorů (Moorův zákon) a zvyšováním frekvence [3]. V posledních letech se však hodnoty frekvence ustálily a razantního zrychlení je dosahováno právě za pomoci paralelizace výpočtu. Paralelizaci můžeme provádět na více jádrových CPU nebo na počítačích spojených v síti.

Pro paralelizaci výpočtu nad CPU slouží nástroj OpenMP. OpenMP je součástí překladačů Fortran a C/C++ a jedná se o standart pro programování počítačů se sdílenou pamětí. K zápisu programu se v OpenMP používají direktivy pro překladač. Program tak může být přeložen pro jeden procesor, nebo použitím příslušných přepínačů pro paralelní jednotky. Nevýhodou výpočtu nad CPU ve většině případech je přítomnost jedné sběrnice, která načítá hodnoty ze společné paměti. Může tak dojít s vysokým počtem procesorů a vysokými požadavky na čtení k její zahlcení. Řešením je zapojení procesorů do sítí a vyvažování čtení z jednotlivých uzlů procesorů. Pro náš problém je však klíčový počet výkonných jednotek a z tohoto důvodu není použití CPU zcela optimální. Další možností jsou grafické karty (GPU).

Označení pro provádění výpočetních operací za pomoci GPU je GPGPU (General-Purpose computation on GPUs). GPU jsou programovatelné paralelní architektury navržené s ohledem na real-time vykreslování grafických primitiv. Současné GPU nabízejí 10x větší propustnost hlavní paměti a používají paralelizaci dat k dosažení až 10x více operací za sekundu oproti současným CPU [6]. Navíc výkonost GPU roste rychleji než Moorův zákon za posledních několik let a tak se výkonový rozdíl mezi GPU a CPU zvětšuje. Tuto skutečnost lze vidět z obrázku 5.1 (převzato z [13]).



Obrázek 5.1: Teoretická rychlost provádění instrukcí v plovoucí řádové čárce na GPU a CPU.

Ještě před několika lety byli programátoři nuceni využít pro výpočty za pomocí GPU čistě grafické primitiva a přístup za pomoci OpenGl či DirectX. To očekávalo od programátorů přetransformování algoritmů tak, aby výpočty odpovídaly grafickým výpočtům poskytovaných těmito knihovnami. Museli se tedy vyrovnat s primární funkcí GPU, převodem 3D modelu scény do obrázku, který je následně vykreslen na obrazovce (více ve vztahu k GPGPU v [14]). Výpočty na grafických kartách se rozšířily do takové podoby, že moderní karty přímo podporují provádění aritmetických operací. Například u karet společnosti ATI jsou vedle možnosti spouštět kernely (malé programy pracující opakovaně na řetězci dat) zpracovávajících vertexy či pixely i výpočetní [1].

Firma nVidia posunula možnosti paralelního přístupu se svým frameworkem CUDA. Tato architektůra umožňuje vyvíjet programy pomocí nejrůznějších jazyků jako jsou například Java, .NET, C, C++, Fortran. Jistou nevýhodou je však dostupnost pouze pro nVidia GPU procesory. Navíc programátor pracující v tomto frameworku se musí orientovat v šesti různých paměťových prostorech, musí mít základní znalost, jakým způsobem jsou vlákna mapovány na GPU hardware a jakým způsobem spolu vlákna komunikují [2]. Framework tak odstínil a usnadnil programování nad GPU jednotkami, ale stále předpokládá od programátora programování jednotlivých jader vykonávajících danou operaci nad daty. To je časově náročné a tento přístup vyžaduje duplicitní zápisy kódu. Tyto nevýhody se snaží odstranit OpenCL.

OpenCL bylo původně vyvinuto firmou Apple (dnes spravována průmyslovým konsorciem Khornos) a na jeho vývoji spolupracuje skupina skládající se z AMD, IBM, Intel a nVidie. Díky spolupráci není OpenCl závislé na platformách a je to tedy framework pro psaní aplikací běžících na CPU, GPU a DSP architektůrách [19]. Podívejme se blíže na OpenCL programový model. Program je spouštěn na výpočetních zařízeních, které se skládají z výpočetních jednotek. Každá výpočetní jednotka je tvořena single-instruction multiple-data procesním elementem (PE). Před spuštěním výpočtu musí všechny elementy

mít obslužný program a nachystána data. Z toho důvodu se nelze při výpočtu obejít bez CPU. Typicky je aplikace spuštěna na CPU a CPU zajistí předpřipravení výpočtu jako je nahrání hodnot do výpočetního zařízení, zajistění distribuce a po skončení kolekci výsledků a jejich vyhodnocení. Datové přenosy mezi jednotkami musí být efektivně naplánovany tak, aby byl minimalizovaný dopad komunikace a musí být rozvrženy s ohledem na lokalitu výpočtů. K optimalizaci datových přenosů nám slouží profilování (viz [12]).

5.2 Příklady zrychlení paralelních systémů

Podívejme se nyní na vybrané problémy, které byly řešeny pomocí paralelních přístupů uvedených výše a spadají do problematiky parciálních diferenciálních rovnic. Jedna z prvních prací zabývající se problematikou GPU řešících parciální diferenciální rovnice byla zaměřena na počítání s matematickými strukturami (matice, vektory) na GPU [17], [15] a řešení problému tepelné rovnice [16].

Řešení advekčně-difúzní rovnice pomocí metody přímky bylo zpracováváno pomocí metody Runge-Kutta čtvrtého řádu [18]. Bylo testováno na CPU Intel Core 2 Duo 2,66GHz a na grafické kartě GeForce 9600M GT a GeForce 8800 Ultra. V průměru bylo dosaženo šesti násobného zrychlení při použití výkonější grafické karty 8800 Ultra a tato karta dokonce přinesla 68,86krát větší rychlost výpočtu než výpočet na CPU. Výpočty nad GPU byly řešeny v rámci frameworku CUDA. V [22] byl problém řešen za pomocí OpenCL na GPU GeForce GT9600M, GeForce GTX285 a ATI HD5870. K procesoru Intel Core 2 Duo 2,66GHz byl problém spuštěn i na Intel Core 2 Quad 2.66GHz. Výsledky v tomto článku lze shrnout tak, že CUDA dosáhla většího zrychlení než OpenCL, ale vždy byl rozdíl v rámci procent, ale zrychlení GPU bylo v desítkách procent oproti CPU.

5.3 Zhodnocení

V této kapitole jsme se seznámili s dostupnými paralelními systémy a trendy jejich vývoje. Jak je vidět, GPU je vhodné použít pro úkoly, které je možné dobře paralelizovat, jako jsou operace nad poli a maticemi prvků. Podmínkou ovšem je, že se nad všemi prvky bude provádět stejná operace a že jednotlivé položky jsou na sobě nezávislé. Také jsme uvedli vědecké články, které se zabývají paralelním výpočtem diferenciálních rovnic.

Prakticky lze s určitými omezeními dostupné paralelní systémy používat, je ale samozřejmé, že speciálně navržená architektura bude rychlejší, ale ekonomicky náročnější. Paralelizace, ať už za použití jakkéhokoliv frameworku, má svou režii, která je nutná na začátku i konci výpočtu. Navíc pro naše řešení je zde potencionální těsná souvislost mezi jednotlivými jednotkami, kdy výsledek z jedné by měl být vstupem do jiné. Synchronizační režie v tomto případě bude hrát podstatnou roli a bude degradovat celkové zrychlení. Neopomenutelným faktem je i nutnost znát detailně problematiku daného frameworku, aby napsané algoritmy využívaly prostředky efektivně. Náš návrh byl optimalizován tak, aby bylo použito co nejméně registrů a tím byly použité prostředky využity s maximální efektivitou.

Kapitola 6

Závěr

V této práci jsme se nejprve seznámili s řešením jednoduchých parciálních rovnic a jejich převodem na soustavu obyčejných diferenciálních rovnic. V řadě řešení se nám objevoval prvek, který plnil funkci integrátoru. Blíže jsme se zaměřili na tento integrátor a navrhli jsme jeho numerickou podobu, která za pomoci Eulerovy metody hledá řešení jen v diskrétních bodech na zvolené množině.

Vzhledem ke značné rozdílnosti reprezentace čísel v pevné a plovoucí řádové čárce jsme řešili návrh numerických integrátoru pro obě reprezentace odděleně. Na základě návrhu numerického integrátoru v pevné řádové čárce jsme navrhli integrátor v plovoucí řádové čárce s ohledem na normu IEEE 754. Srovnáme-li tyto dva návrhy, je jasné, že integrátor pracující v plovoucí řádové čárce je složitější z důvodu zpracování postupně mantisy a exponentu. Jeho obvodová realizace tak obsahuje více registrů díky přítomnosti registrů pomocných. Mnohem větší přesnost integrátoru pracující v plovoucí řádové čárce je tak vykoupena větší plochou na případném čipu. Musíme si však uvědomit, že přesnost je od integrátoru očekávána a že dosažení větší přesnosti u integrátoru v pevné řádové čárce bude mít vliv na velikosti všech registrů. Zvětšování přesnosti však v tomto případě nebude mít vliv na zvětšení komunikačních propojení v případě, kdy budeme chtít zapojit více numerických integrátorů. A to v obou realizacích. Je to dáno tím, že procesory dokáži spolu komunikovat a předávat si data za pomoci jednobitového vodiče.

Díky analýze pohybu dat v sériovém numerickém integrátoru jsme mohli analyzovat paralelní operace a zrychlit tak výpočet. Dostali jsme numerický integrátor, který zpracovává mantisu a exponent paralelně. Pro oba návrhy jsme vytvořili simulátor, který simuluje výpočet dle navržených algoritmů. Zhodnotili jsme tak, že paralelním přístupem dosahujeme zrychlení, které se však potýká s velkou synchronizační režií a s čím většími bitovými hodnotami pracujeme, tím se řešení blíží k rychlosti postupného zpracování. Navíc i množství nutných obvodů dosahuje dvojnásobného množství, čímž se zvedají požadavky na plochu případné hardwarové realizace.

Řešení postavené na Eulerově metodě nepřináší uspokojivou přesnost numerického výpočtu. Pro paralelní i sériový přístup jsme tak navrhli modifikace, pomocí kterých lze počítat více členů Taylorova rozvoje a tím získat i přesnější výsledky. Tyto modifikace nebyly nikterak složité a zachovaly původní koncepce návrhu. Razantní zvýšení přesnosti tak lze dosáhnout malými úpravami návrhu postaveného na Eulerově metodě.

Současné paralelní přístupy jsou velmi využívané pro operace SIMD. Tento přístup je už z principu v grafických kartách, které tak dominují v paralelních výpočtech díky své ceně a množství vývojových nástrojů usnadňujících psaní programů. Univerzální použití GPU pro paralelní výpočet přináší zpomalení v podobě inicializačních kroků a řídící režie. Velmi tak

záleží na konkrétní aplikaci, která předurčuje dosažené zrychlení oproti referenčnímu sériovému přístupu. Náš přístup specializovaného numerického integrátoru byl navržen s ohledem na minimalizaci počtu jednotek, na rychlost zpracování a efektivní využití prostoru.

Některé návrhy uvedené v této práci se již využívají ve výuce na Fakultě informačních technologií VUT v Brně. Jedná se především o návrhy postavené na principu Eulerovy metody, které jsou zařazeny do výuky v rámci předmětu IPR bakalářského studijního programu a magisterského kurzu VNV. Pro studijní účely byly také vytvořeny podpůrné prezentace popisující činnost numerických integrátorů na konkrétním příkladu. Vytvořená aplikace byla navržena s ohledem na názornost a popis algoritmů. Může tak být zařazena do výuky pro studenty těchto předmětů jako příklad aplikace a výpočtů probíraného učiva.

Literatura

[1] ATI: *ATI Stream Computing OpenCL Programming Guide*. 2010 [cit. 2011-04-05]. URL http://developer.amd.com/gpu_assets/ATI_Stream_SDK_OpenCL_Programming_Guide.pdf

[2] Bakkum, P.; Skadron, K.: Accelerating SQL database operations on a GPU with CUDA. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-935-0, s. 94–103, doi:http://doi.acm.org/10.1145/1735688.1735706.

[3] Brandvik, T.; Pullan, G.: Acceleration of a 3D Euler solver using commodity graphics hardware. In *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008.

[4] Drábek, V.: *Výstavba počítačů*, ročník 1. Brno: Opora. Fakulta informačních technologií, 1996, 150 s.

[5] Fučík, O.: *Návrh číslicových systémů INC*. Brno: Opora. Fakulta informačních technologií, 2006.

[6] Govindaraju, N. K.; Larsen, S.; Gray, J.; aj.: A memory model for scientific algorithms on graphics processors. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA: ACM, 2006, ISBN 0-7695-2700-0, doi:http://doi.acm.org/10.1145/1188455.1188549.

[7] Kahan, W.: *IEEE Standard 754 for Binary Floating-Point Arithmetics*. University of Califormia, 1996-06-31.

[8] Koren, I.: *Computer arithmetic algorithms*, ročník 2. A K Peters, Ltd., 2002, ISBN 1-56881-160-8, 281 s.

[9] Kunovský, J.: *Modern Taylor Series Method*. FEI-VUT Brno, 1994, habilitation work.

[10] Kunovský, J.: *Prvky počítačů IPR*. Opora. FIT VUT v Brně, 2006.

[11] M. Kubíček, D. J., M. Dubcová: *Numerické metody a algoritmy*. Vydavatelství VŠCHT Praha, 2005, ISBN 80-7080-558-7.

[12] Mistry, P.; Gregg, C.; Rubin, N.; aj.: Analyzing program flow within a many-kernel OpenCL application. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, GPGPU-4, New York, NY, USA: ACM, 2011, ISBN 978-1-4503-0569-3, s. 10:1–10:8, doi:http://doi.acm.org/10.1145/1964179.1964193.

[13] Nvidia: *NVIDIA CUDA C Programing Guide*. 2010 [cit. 2011-04-09].
URL http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf

[14] Owens, J. D.; Luebke, D.; Govindaraju, N.; aj.: A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, ročník 26, č. 1, Březen 2007: s. 80–113, doi:10.1111/j.1467-8659.2007.01012.x.

[15] Rumpf, M.; Strzodka, R.: Level set segmentation in graphics hardware. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, ročník 3, 2001, s. 1103 – 1106 vol.3, doi:10.1109/ICIP.2001.958320.

[16] Rumpf, M.; Strzodka, R.: Using Graphics Cards for Quantized FEM Computations. In *Proceedings of IASTED Visualization, Imaging and Image Processing Conference (VIIP'01)*, 2001, s. 193–202.

[17] Rumpf, M.; Strzodka, R.; Bonn, U.: Nonlinear Diffusion in Graphics Hardware. In *In Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '01*, Springer, 2001, s. 75 – 84.

[18] Simek, V.; Dvorak, R.; Zboril, F.; aj.: GPU Accelerated Solver of Time-Dependent Air Pollutant Transport Equations. *Digital Systems Design, Euromicro Symposium on*, ročník 0, 2009: s. 707–713, doi:http://doi.ieeecomputersociety.org/10.1109/DSD.2009.146.

[19] Stone, J.; Gohara, D.; Shi, G.: OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science Engineering*, ročník 12, č. 3, may-june 2010: s. 66 –73, ISSN 1521-9615, doi:10.1109/MCSE.2010.69.

[20] Tišňovský, P.: Fixed point arithmetic [online]. `http://www.root.cz/clanky/fixed-point-arithmetic/`, 2006-05-24 [cit. 2010-11-10].

[21] Tišňovský, P.: Aritmetické operace s hodnotami ve formátu plovoucí řádové čárky [online]. `http://www.root.cz/clanky/aritmeticke-operace-s-hodnotami-ve-formatu-plovouci-radove-carky/`, 2006-06-07 [cit. 2010-11-10].

[22] Šimek, V.; Dvořák, R.; Zbořil, V. F.; aj.: Performance Evaluation of OpenCL Framework for Numerical Solver of Advection Diffusion Equation. In *Proceedings of CSE 2010 International Scientific Conference on Computer Science and Engineering*, The University of Technology Košice, 2010, ISBN 978-80-8086-164-3, s. 279–286.

Dodatek A

Obsah CD

Přiložené CD obsahuje následující materiál:

/text/ - zdrojové soubory této práce v LaTeXu

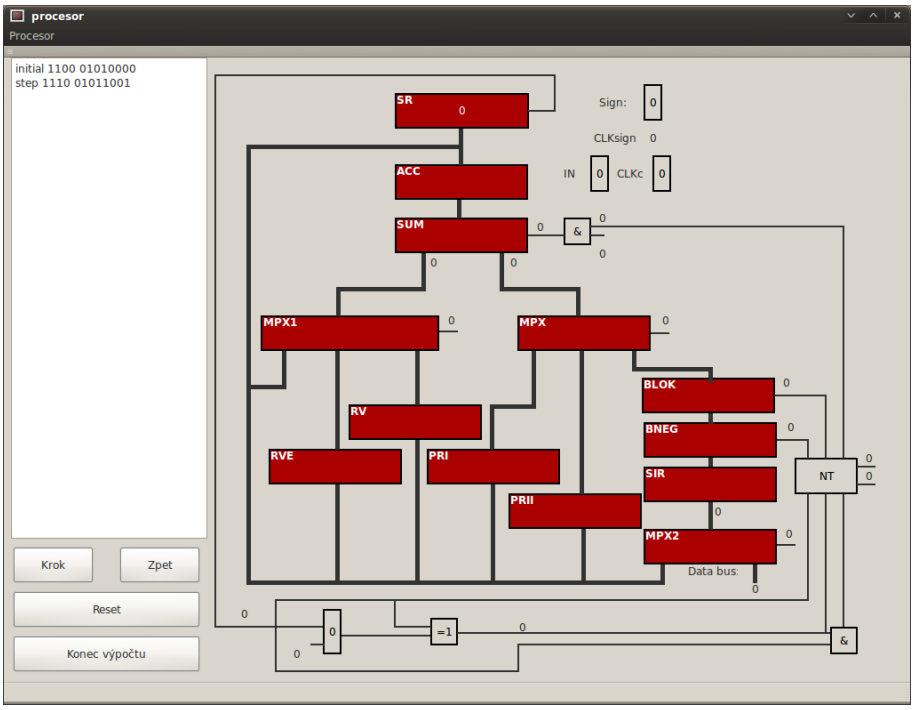
/simulator/ - zdrojové soubory aplikace simulující činnost numerických integrátorů postavených na principu Eulerovy metody

/statistiky/ - soubory obsahující statistické přehledy získané z aplikace, jejichž výsledky jsou zahrnuty v této práci

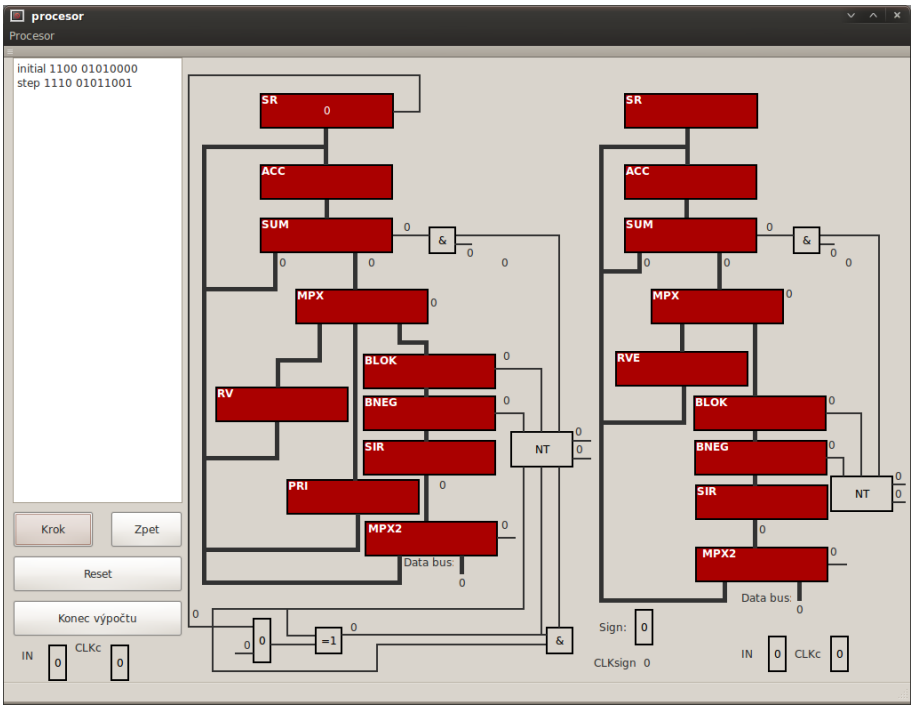
/prezentace/ - Microsoft PowerPoint prezentace činnosti numerických integrátorů na zvoleném příkladu

Dodatek B

Rozložení aplikace



Obrázek B.1: Numerický integrátor postavený na principu Eulerovy metody zpracovávající mantisu a exponent postupně.



Obrázek B.2: Numerický integrátor postavený na principu Eulerovy metody zpracovávající mantisu a exponent paralelně.

Dodatek C

Přehled operací

Příklad	Y	H	$Y_1 = Y + H * Y$
1.	0 1110 01011001	0 0001 01010000	0 1111 01100011
2.	0 0001 01010000	0 1110 01011001	0 0001 01001011
3.	0 1100 01010000	0 1110 01011001	0 1100 01011101
4.	0 1110 01011001	0 1101 00100000	0 1110 01011011
5.	0 1101 00100000	0 1110 01011001	0 1101 00100101
6.	0 11101010 0101100010010010	0 11100010 0001000100011000	0 11101010 0101100010010010
7.	0 00000110 0101100010010010	0 00000100 0001000100011000	0 00001010 0001000101011100
8.	0 00000110 0001001010010010	0 00010100 0011100000011000	0 00011010 0000100000100011
9.	0 11100110 0001001010010010	0 11100100 0011100000011000	0 11100110 0001001010010010
10.	0 11000001 0001000111000000	0 11001101 0010101010101010	0 11000001 0001000111000000

Tabulka C.1: Vstupní operandy, výsledky numerické integrace.

C.1 Numerický integrátor zpracovávající mantisu a exponent ve společné jednotce

	1. příklad	2. příklad	3. příklad	4. příklad	5. příklad
Počet instrukcí					
move	70	71	71	72	71
load	9	9	9	9	9
mem	4	4	4	4	4
shl	0	0	0	0	0
shr	9	10	10	11	10
sha	7	7	7	7	7
inc	7	6	7	9	6
rst	12	16	14	12	16
sum	26	27	27	28	27
nop	0	0	0	0	0
Vodiče					
getSignal	1697	1748	1747	1800	1748
setSignal	115	115	115	115	115
getWire	0	0	0	0	0
setWire	477	489	489	501	489
Logické operace					
EXOR	159	163	163	167	163
XNOR	37	40	40	43	40
AND	318	326	326	334	326
OR	477	489	489	501	489
Počet změn hodnot v registrech					
SIR	9	9	9	9	9
BNEG	17	15	15	15	15
BLOK	17	15	15	15	15
ACC	22	22	22	23	22
SR	1	1	1	1	1
RVE	2	2	2	2	2
RV	1	1	1	1	1
PRI	2	2	2	2	2
PRII	1	1	1	1	1

Tabulka C.2: Výstupní hodnoty simulace - krátké operandy.

	6. příklad	7. příklad	8. příklad	9. příklad	10. příklad
Počet instrukcí					
move	123	97	113	121	144
load	9	9	9	9	9
mem	4	4	4	4	4
shl	0	0	0	0	0
shr	46	20	36	44	67
sha	15	15	15	15	15
inc	38	12	28	36	56
rst	16	16	16	16	22
sum	63	37	53	61	84
nop()	0	0	0	0	0
Vodiče					
getSignal	3752	2353	3217	3644	4889
setSignal	179	179	179	179	179
getWire	0	0	0	0	0
setWire	969	657	849	945	1221
Logické operace					
EXOR	323	219	283	315	407
XNOR	124	46	94	118	187
AND	646	438	566	630	814
OR	969	657	849	945	1221
Počet změn hodnot v registrech					
SIR	9	9	9	9	9
BNEG	23	25	25	23	23
BLOK	23	25	25	23	23
ACC	58	33	49	56	79
SR	1	1	1	1	1
RVE	2	2	2	2	2
RV	1	1	1	1	1
PRI	2	2	2	2	2
PRII	1	1	1	1	1

Tabulka C.3: Výstupní hodnoty simulace - delší operandy.

C.2 Numerický integrátor zpracovávající mantisu a exponent v oddělených jednotkách

C.2.1 Jednotka mantisy

	1. příklad	2. příklad	3. příklad	4. příklad	5. příklad
Počet instrukcí					
move	37	37	37	37	37
load	4	4	4	4	4
mem	2	2	2	2	2
shl	0	0	0	0	0
shr	8	8	8	8	8
sha	7	7	7	7	7
inc	3	1	2	3	1
rst	6	10	8	6	10
sum	14	14	14	14	14
nop()	6	10	10	14	10
Vodiče					
getSignal	746	776	775	802	776
setSignal	83	83	83	83	83
getWire	0	0	0	0	0
setWire	270	282	282	294	282
Logické operace					
EXOR	90	94	94	98	94
XNOR	10	10	10	10	10
AND	180	188	188	196	188
OR	270	282	282	294	282
Počet změn hodnot v registrech					
SIR	4	4	4	4	4
BNEG	10	10	10	10	10
BLOK	10	10	10	10	10
ACC	12	12	12	12	12
SR	1	1	1	1	1
RV	1	1	1	1	1
PRI	1	1	1	1	1

Tabulka C.4: Výstupní hodnoty simulace jednotky mantisy - krátké operandy.

	6. příklad	7. příklad	8. příklad	9. příklad	10. příklad
Počet instrukcí					
move	61	61	61	61	61
load	4	4	4	4	4
mem	2	2	2	2	2
shl	0	0	0	0	0
shr	16	16	16	16	16
sha	15	15	15	15	15
inc	5	5	5	5	2
rst	10	10	10	10	16
sum	22	22	22	22	22
nop()	122	18	82	114	206
Vodiče					
getSignal	1956	1228	1676	1900	2547
setSignal	147	147	147	147	147
getWire	0	0	0	0	0
setWire	762	450	642	738	1014
Logické operace					
EXOR	254	150	214	246	338
XNOR	10	10	10	10	10
AND	508	300	428	492	676
OR	762	450	642	738	1014
Počet změn hodnot v registrech					
SIR	4	4	4	4	4
BNEG	18	18	18	18	18
BLOK	18	18	18	18	18
ACC	20	20	20	20	20
SR	1	1	1	1	1
RV	1	1	1	1	1
PRI	1	1	1	1	1

Tabulka C.5: Výstupní hodnoty simulace jednotky mantisy - delší operandy.

C.2.2 Jednotka exponentu

	1. příklad	2. příklad	3. příklad	4. příklad	5. příklad
Počet instrukcí					
move	23	24	24	25	24
load	4	4	4	4	4
mem	2	2	2	2	2
shl	0	0	0	0	0
shr	1	2	2	3	2
sha	0	0	0	0	0
inc	3	5	5	6	5
rst	7	7	7	7	7
sum	9	10	10	11	10
nop()	25	25	25	25	25
Vodiče					
getSignal	642	687	687	729	687
setSignal	27	27	27	27	27
getWire	0	0	0	0	0
setWire	270	282	282	294	282
Logické operace					
XNOR	13	17	17	20	17
AND	90	94	94	98	94
Počet změn hodnot v registrech					
SIR	4	4	4	4	4
BNEG	5	5	5	5	5
BLOK	5	5	5	5	5
ACC	8	9	9	10	9
SR	0	0	0	0	0
RVE	2	2	2	2	2

Tabulka C.6: Výstupní hodnoty simulace jednotky exponentu - krátké operandy.

	6. příklad	7. příklad	8. příklad	9. příklad	10. příklad
Počet instrukcí					
move	52	26	42	50	73
load	4	4	4	4	4
mem	2	2	2	2	2
shl	0	0	0	0	0
shr	30	4	20	28	51
sha	0	0	0	0	0
inc	33	6	22	31	54
rst	7	7	7	7	7
sum	38	12	28	36	59
nop()	73	73	73	73	73
Vodiče					
getSignal	2151	1056	1728	2067	3033
setSignal	27	27	27	27	27
getWire	0	0	0	0	0
setWire	762	450	642	738	1014
Logické operace					
XNOR	101	22	70	95	164
AND	254	150	214	246	338
Počet změn hodnot v registrech					
SIR	4	4	4	4	4
BNEG	5	5	5	5	5
BLOK	5	5	5	5	5
ACC	37	11	27	35	58
SR	0	0	0	0	0
RVE	2	2	2	2	2

Tabulka C.7: Výstupní hodnoty simulace jednotky exponentu - delší operandy.